WEST Search History

```
DATE:  Thursday, August 21, 2003
```

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
DB=USPT,PGPB; PLUR=YES; OP=ADJ			
L9	L4 and (add$ near device)	3026	L9
L8	L4 and (add$ near device)	2796	L8
L7	L6 and DDE	7	L7
L6	L5 and PCMS	694	L6
L5	L4 and device	50650	L5
L4	power near (control or management)	63153	L4
L3	l1 and power near control	1	L3
DB=JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L2	power builder or powerbuilder	5	L2
DB=USPT,PGPB; PLUR=YES; OP=ADJ			
L1	power builder or powerbuilder	205	L1

```
END OF SEARCH HISTORY
```

WEST Search History

DATE: Thursday, August 21, 2003

Set Name side by side	Query	Hit Count	Set Name result set
DB=JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L10	L4 and DDE	1	L10
DB=USPT; PLUR=YES; OP=ADJ			
L9	L1 and DDE	21	L9
L8	L2 and intelligent	9	L8
L7	L2 and DDE	0	L7
DB=JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L6	L4 and (adding near device)	11	L6
L5	L4 and (power?builder or PowerBuilder or power builder)	0	L5
L4	power near (management or control)	39722	L4
DB=USPT,PGPB; PLUR=YES; OP=ADJ			
L3	L2 and (power?builder or PowerBuilder or power builder)	0	L3
L2	L1 and (adding near device)	112	L2
L1	power near (management or control)	63153	L1

END OF SEARCH HISTORY

> home > about > feedback > login
US Patent & Trademark Office

Try the *new* Portal design
Give us your opinion after using it.

Search Results

Search Results for: **[Powerbuilder]**
Found **12** of **120,398 searched.**

Search within Results

> Advanced Search
> Search Help/Tips

Sort by: Title Publication Publication Date Score Binder

Results 1 - 12 of 12 short listing

1 Consumer-centric reengineering at the Colorado Department of Revenue 82%
Anol Bhattacherjee
Communications of the AIS June 2000

2 Recruiting and retention of information systems professionals in Nebraska: issues and challenges 80%
Mike Echols , Uma G. Gupta
Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research June 1998

3 Experience teaching an introduction to DBMS 77%
Mario A. M. Guimaraes
ACM SIGCSE Bulletin , Working group reports from ITiCSE on Innovation and technology in computer science education December 1999
Volume 31 Issue 4
This paper describes the methodology used to teach an introductory Database Management System course. Although all content description refers exclusively to this class, the methodology can be used to teach other computer science courses. There are several unique aspects of the course. Students do a preliminary presentation and a final presentation of their projects. During the preliminary presentation, they perform the role of the client (inversion of roles); while during the final presentation, ...

4 Skill requirements of IT&T professionals and graduates: an Australian study research-in-progress 77%
Jo Orr , Liisa von Hellens
Proceedings of the 2000 ACM SIGCPR conference on Computer personnel research April 2000
The IT&T (Information Technology and Telecommunications) Skills Taskforce recently

released the results of their study on IT&T skill requirements in Australia, predicting that in the current financial year, Australian employers will be seeking an excess of 30,000 skilled IT&T professionals [8]. This figure is expected to grow at approximately 9% per year for at least the next 5 years [8]. Yet already, employers are reported to be experiencing difficulty obtaining skilled IT& ...

5 Technical opinion: reuse: been there, done that 77%
Jeffrey S. Poulin
Communications of the ACM May 1999
Volume 42 Issue 5

6 Introducing client/server technologies in information systems curricula 77%
Abhijit Chaudhury , H. Raghav Rao
ACM SIGMIS Database September 1997
Volume 28 Issue 4
One goal of information systems (IS) departments in business schools is to train IS professionals with the necessary technical skills to support the IS function in companies. This paper suggests that changes are needed for most current IS curricula to meet the technical requirements of the client/server (C/S) world of technologies. It is hoped that the ideas presented here will stimulate debate and discussions as to how this transition can be accomplished.

7 Forum: Cobol in question 77%
Diane Crawford
Communications of the ACM December 1997
Volume 40 Issue 12

8 The IBM data warehouse architecture 77%
Charles Bontempo , George Zagelow
Communications of the ACM September 1998
Volume 41 Issue 9

9 Reuse research and development: is it on the right track? 77%
Mansour Zand , Gillermo Arango , Maggie Davis , Ralph Johnson , Jeffrey S. Poulin , Andrew Watson
ACM SIGSOFT Software Engineering Notes , Proceedings of the 1997 symposium on Software reusability May 1997
Volume 22 Issue 3

10 Employment outsourcing in information systems 77%
Sandra Slaughter , Soon Ang
Communications of the ACM July 1996
Volume 39 Issue 7

11 New wave prototyping: use and abuse of vacuous prototypes 77%
Hal Berghel

interactions April 1994
Volume 1 Issue 2

12 Smalltalk in the business world (panel): the good, the bad, and the future 77%
Yen-Ping Shan , Ken Auer , Andrew J. Bear , Jim Adamczyk , Adele Goldberg , Tom Love , Dave Thomas
ACM SIGPLAN Notices , Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications October 1994
Volume 29 Issue 10

Results 1 - 12 of 12 short listing

Chapter 22

Creating Executables

About this chapter

This chapter describes how to create an executable version of your target.

About building Pocket PowerBuilder targets

Using the Project painter

You use the Project painter to create an executable version of your target that you can deploy to Windows CE devices, emulators, or the desktop. The Project painter allows you to streamline the generation of the files your target needs and to rebuild the target easily after you make changes to target objects.

For how to create a new project, see "Creating a project".

Building executable files

If you are building an executable file, there are two basic ways to package the application:

- As one standalone executable file that contains all the objects in the application
- As an executable file and one or more dynamic libraries that contain objects that are linked at runtime

Read the chapter on packaging your application for deployment in the *Resource Guide and Reference* to get an understanding of the best way for you to package the application. Then follow the procedures in "Defining a project" to implement your strategy.

Providing other resources

You might need to provide additional resources that your target uses, such as bitmaps and icons. There are two ways to provide resources:

- Distribute them separately
- Include them in a Pocket PowerBuilder resource file (*PKR*) and build an executable or a dynamic library using the resource file

For more information, see "Distributing resources ".

Building the workspace

You can build and deploy all the targets in your workspace using buttons on the PowerBar, pop-up menus in the System Tree, or a command line. For more information, see the *Resource Guide and Reference*.

Creating a project

You can create a new project when you create a new template application. At any other time, you can do this from the Project page in the New dialog box.

The Project page has two icons: one that opens a wizard to help you set up a project, and another that opens only the Project painter, where you can enter the required information without being prompted. The following procedure describes how to create a new project from the Project page.

To create a new project object from the Project page:

1. **Select File>New or click the New button in the PowerBar to open the New dialog box.**
2. **Select the Project tab.**
3. **Select the Application Wizard or Application icon and click OK.**
   - If you selected the Application wizard, complete the wizard screens to create a new project with the properties for which you are prompted, and then click Finish

     After you click Finish, the Project painter opens and displays your selections. You can also open the project generated by the wizard at any time to modify the properties you selected and build the project.
   - If you selected the Application icon, the Project painter opens so that you can specify properties of the project object

     After you close the Project painter, you can reopen the project from the System Tree at any time, modify the properties you selected, and deploy the project to the platforms you select.

Defining a project

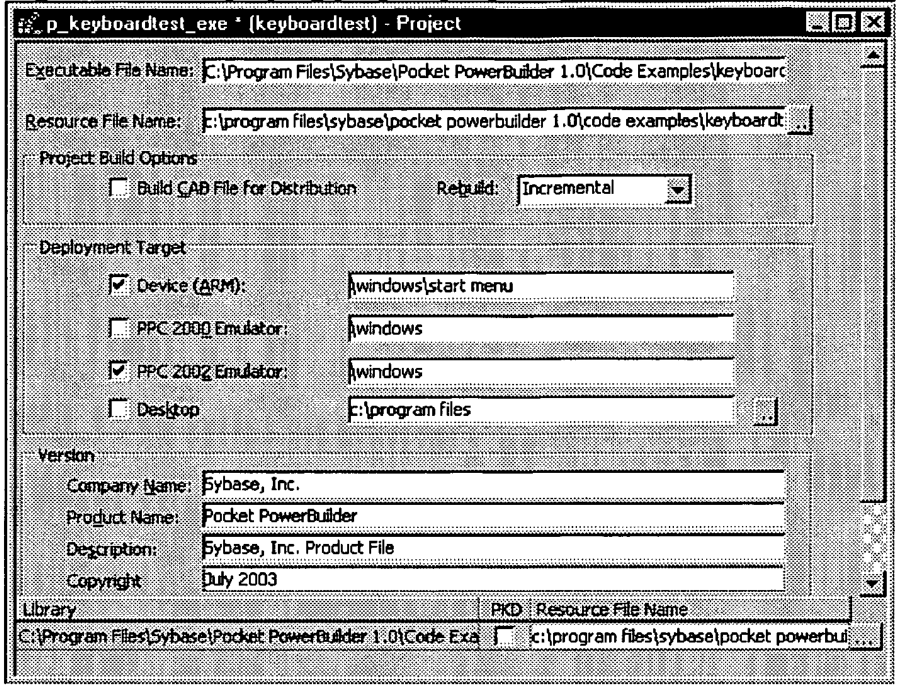
The Project painter for executable applications allows you to streamline the generation of executable files and dynamic libraries. When you build a project object, you specify the following components of your application:

- Executable file name
- Which of the libraries you want to distribute as dynamic libraries
- Which Pocket PowerBuilder resource files (if any) should be used to build the executable file and the dynamic libraries

- Which build options you want to use in your project
- Which platforms you want to deploy to and what directories you want to use as your deployment directories
-

Version information for your application

Figure 22-1: Displaying project properties in the Project painter



p_keyboardtest_exe * (keyboardtest) - Project

Executable File Name: C:\Program Files\Sybase\Pocket PowerBuilder 1.0\Code Examples\keyboarc

Resource File Name: c:\program files\sybase\pocket powerbuilder 1.0\code examples\keyboardt

Project Build Options

[ ] Build CAB File for Distribution   Rebuild: Incremental

Deployment Target

[✓] Device (ARM): \windows\start menu

[ ] PPC 2000 Emulator: \windows

[✓] PPC 2002 Emulator: \windows

[ ] Desktop: c:\program files

Version

Company Name: Sybase, Inc.

Product Name: Pocket PowerBuilder

Description: Sybase, Inc. Product File

Copyright: July 2003

Library	PKD	Resource File Name
C:\Program Files\Sybase\Pocket PowerBuilder 1.0\Code Exa	[ ]	c:\program files\sybase\pocket powerbui

After you create a project, you might need to update it because your library list has changed or because you want to change your compilation options.

To define or modify a project:

1. **Open a project in the Project painter by:**

   - Creating a new application project

     For information on creating a project, see "Creating a project".

   - Selecting File>Open from the Pocket PowerBuilder menu, then selecting the target, the library, the object type (Project), and the existing project that you want to modify in the object list box

   - Double-clicking a project in the System Tree

   The Project painter workspace displays.

2. **Specify or modify options as needed.**

If you opened an existing project or a project created using the wizard, previously selected options display in the workspace. For information about each option, see "Executable application project options".

3. **When you have finished defining the project object, save the object by selecting File>Save from the menu bar.**

   Pocket PowerBuilder saves the project as an independent object in the specified library. Like other objects, projects are listed in the System Tree and the Library painter.

Executable application project options

Table 22-1 describes each of the options you can specify in the Project painter for executable applications. You can also specify most of these options in the Application Project wizard.

Table 22-1: Project options

Option	What you specify or select
Executable File Name	Specify a name for the executable. The name must have the extension *EXE*. If you do not want the executable saved to your current directory, click the Browse (...) button next to the box to navigate to a different directory.
Resource File Name	(Optional) Specify a Pocket PowerBuilder resource file (PKR) for your executable if you dynamically reference resources (such as bitmaps and icons) in your scripts and you want the resources included in the executable file instead of having to distribute them separately. You can type the name of a resource file in the box or click the button next to the box to browse your directories for the resource file you want to include. For more information, see "Distributing resources ".
Build CAB File for Distribution	Select this if you want Pocket PowerBuilder to build a CAB file for deployment to Windows CE devices. For more information about building and distributing CAB files, see the chapter on packaging an application for distribution in the *Resource Guide and Reference*.
Rebuild	Specify Full to have Pocket PowerBuilder regenerate all objects in the application libraries before it creates the executable and dynamic libraries. Specify Incremental if you want Pocket PowerBuilder to regenerate only objects that have changed--and objects that reference any objects that have changed--since the last time you built your application.
Device (ARM)	Select to deploy the project to a device of type ARM. Enter the directory on the device where you want to deploy the project. The default deployment directory is *\Program Files*. You might consider changing the deployment directory to a directory you create in the Start Menu path. For more information, see Appendix B, "PowerBuilder and Pocket PowerBuilder Product Differences".
PPC 2000 Emulator	Select if you want to deploy the project to the emulator. Enter the directory on the emulator where you want to deploy the project. The default deployment directory is *\Program Files*.
	Select if you want to deploy the project to the emulator. Enter the

PPC 2002 Emulator	directory on the emulator where you want to deploy the project. The default deployment directory is *\Program Files*.
Desktop	Select if you want to deploy the project to the desktop. Enter the directory on the desktop where you want to deploy the project. You can use the browse button next to the desktop directory box to browse for a directory.
Version	Specify your own values for the Company Name, Product Name, Description, Copyright, and Version fields associated with the executable file. These values become part of the Version resource associated with the executable file, and--if you deploy to the desktop--most of these values display on the Version tab page of the Properties dialog box for the file in Windows Explorer. The Company Name and Product Name fields are visible in the Remove Programs list on Windows CE platforms, but only if you use CAB files to distribute your projects.
PKD	Select this check box to define a library as a dynamic library to be distributed with your application.
Resource File Name	Specify a resource file for a dynamic library if library objects use resources (such as bitmaps and icons) and you want to include the resources in the dynamic library instead of having to distribute them separately. The file name cannot be specified in the wizard.

Using dynamic libraries

About dynamic libraries

You can store the objects used in your Pocket PowerBuilder application in more than one library and, when you run the application, dynamically load any objects that are not contained in the application's executable file. This allows you to break the application into smaller units that are easier to manage and makes the executable file smaller. You do this by using dynamic libraries. Pocket PowerBuilder builds Pocket PowerBuilder dynamic libraries (PKD files).

Pocket PowerBuilder dynamic libraries are given the name of the PKL with the extension *PKD*. For example, the dynamic library built from *test.pkl* is named *test.pkd*.

Reducing the size of dynamic libraries

When Pocket PowerBuilder builds a dynamic library, it copies the compiled versions of all objects from the source library (PKL file) into the dynamic library.

The easiest way to specify source libraries is simply to use your standard Pocket PowerBuilder libraries as source libraries. However, using this technique can make your dynamic libraries larger than they need to be, because they include all objects from the source library, not just the ones used in your application. You can create a Pocket PowerBuilder library that contains only the objects that you want in a dynamic library.

To create a source library to be used as a dynamic library:

1. **In the Library painter, place in one standard Pocket PowerBuilder library (a PKL file) all the**

objects that you want in the dynamic library.

If you need to create a new library, select Entry>Library>Create from the menu bar, then drag or move the objects into the new library.

2. **Make sure the application's library search path includes the new library.**

Multiple dynamic libraries

You can use as many dynamic libraries as you want in an application. To do so, create a source library (PKL file) for each of them.

Specifying the dynamic libraries in your project

When you define your project, you tell Pocket PowerBuilder which of the libraries in the application's library search path will be dynamic by checking the PKD check box next to the library name in the Project painter.

Including additional resources for a dynamic library

When building a dynamic library, Pocket PowerBuilder does not inspect the objects; it simply copies the compiled form of the objects into the dynamic library. Therefore, if any of the objects in the library use resources (pictures and icons)--either specified in a painter or assigned dynamically in a script--and you do not want to provide these resources separately, you must list the resources in a PKR file. Doing so enables Pocket PowerBuilder to include the resources in the dynamic library when it builds it.

To reference additional resources:

1. **List the resources in a PKR file, as described in "Using Pocket PowerBuilder resource files".**

2. **Use the Resource File Name box in the Project painter workspace to reference the PKR file in the dynamic library.**

Distributing resources

You can choose to distribute your resources (pictures and icons) separately or include them in your executable file or dynamic library.

Distributing resources separately

When a resource is referenced at runtime and it has not been included in the executable file or in a dynamic library, Pocket PowerBuilder looks for it in the device path provided for the resource. In test mode, you need only make sure the referenced files are in your machine's search path, but in applications that you deploy to a Windows CE device, you must either include the full path to the location where you install the referenced files or install the resource files in the default *Windows* directory.

For example, if you include picture files in script, you can use a full path name for the device where they will be installed:

```
IF Balance < 0 THEN
   p_logo.PictureName = &

    "\program files\sybase\frown.bmp"
ELSE
   p_logo.PictureName = "smile.bmp"
END IF
```

You can distribute the files *frown.bmp* and *smile.bmp* with your application. You must install the *frown.bmp* file to the *\Program Files\Sybase* directory on the Windows CE device, and the *smile.bmp* file to the *\Windows* directory.

In test mode on the desktop, if the *smile.bmp* file is in the search path at runtime, the application can load it as needed. The desktop search path is as follows: current directory, Windows directory, Windows System directory, then all directories in the PATH environment variable.

Using Pocket PowerBuilder resource files

Instead of distributing resources separately, you can create a PKR file that lists all dynamically assigned resources.

A PKR file is a Unicode or ANSI text file in which you list resource names (such as BMP, ICO, and GIF files) and DataWindow objects. To create a PKR file, use a text editor. List the name of each resource, one resource on each line, then save the list as a file with the extension *PKR*. Here are the contents of a sample PKR file:

```
ct_graph.ico
document.ico
codes.ico
button.bmp
next1.bmp
prior1.bmp
```

Pocket PowerBuilder compiles the listed resources into the executable file or a dynamic library file, so the resources are available directly at runtime.

Using DataWindow objects

If the objects in one PKL reference DataWindow objects (either statically or dynamically) that are in a different PKL, you must either specify a Pocket PowerBuilder resource file that includes the DataWindow objects, or define the library that includes them as a PKD that you distribute with your application. Unlike image files, you cannot distribute the objects separately.

For more information about creating and using PKR files, see the chapter on packaging an application for distribution in the *Resource Guide and Reference*.

What happens at runtime

When a resource such as a bitmap is referenced at runtime, Pocket PowerBuilder first looks in the executable file for it. Failing that, it looks in the PKD files that are defined for the application. Failing that, in test mode only (that is, on the desktop), it looks in directories in the search path for the file.

Building a project

Once you have completed development and defined your project, you build the project to create the executable files and all specified dynamic libraries. You can build your project whenever you have made changes to the objects and want to test or deploy another version of your application.

This section describes building a single project in the Project painter. You can build all the targets in your workspace at any time using buttons on the PowerBar, pop-up menus in the System Tree, or a command line. For more information, see the section on building workspaces in the *Introduction to Pocket PowerBuilder*.

To build a project:

1. **Open a project you built in the Project painter.**

2. **Click the Build button in the PainterBar, or select Design>Build Project.**

   If the target's library list has changed

   When you click Build, Pocket PowerBuilder checks your target's library list. If it has changed since you defined your project, Pocket PowerBuilder updates the Project painter workspace with the new library list. Make whatever changes you need in the workspace, then save the project and click Build again.

   Pocket PowerBuilder builds the executable and all specified dynamic libraries.

The next two sections describe in detail how Pocket PowerBuilder builds the project and finds the objects used in the target.

When Pocket PowerBuilder has built the target, you can check which objects are included in the target. See "Listing the objects in a project".

How Pocket PowerBuilder builds the project

When Pocket PowerBuilder builds your application project:

1. If you selected "Rebuild: Full", Pocket PowerBuilder regenerates all the objects in the libraries.

2. If you selected "Build CAB File for Distribution", Pocket PowerBuilder creates a CAB file for the deployment targets you selected.

   In the process of building a CAB file, Pocket PowerBuilder also creates INF, BAT, DAT, and LOG files. For a detailed description of what gets generated, see the chapter on packaging an application for distribution in the *Resource Guide and Reference*. For a discussion of what to do if errors occur during CAB file generation, see "Troubleshooting errors during CAB file generation".

3. To create the executable file you specified, Pocket PowerBuilder searches through your target. It copies--into the executable file--the compiled versions of referenced objects from the libraries in the target's library search path that are not specified as dynamic libraries. For more details, see "How

Pocket PowerBuilder searches for objects".

4. Pocket PowerBuilder creates a dynamic library for each of the libraries you specified for the target and maintains a list of these library files. Pocket PowerBuilder maintains the unqualified file names of the dynamic library files; it does not save the path name.

Pocket PowerBuilder does not copy objects that are not referenced in the application to the executable file, nor does it copy objects to the executable file from libraries you declared to be dynamic libraries. These objects are linked to the target at runtime and are not stored in the executable file.

What happens during execution

When an object such as a window is referenced in the application, Pocket PowerBuilder first looks in the executable file for the object. If it does not find it there, it looks in the dynamic library files that are defined for the target. For example, if you specified that a dynamic library should be generated from *test.pkl*, Pocket PowerBuilder looks for *test.pkd* at runtime.

The dynamic library files must be in the *\Windows* path on the device or in the directory where you deploy the project EXE. If Pocket PowerBuilder cannot find the object in any of the dynamic library files, it reports a runtime error.

How Pocket PowerBuilder searches for objects

When it searches through a target, Pocket PowerBuilder does not find all the objects that are used in your target and copy them to the executable file. This section describes which objects it finds and copies, and which it does not.

Which objects are copied to the executable file

Pocket PowerBuilder finds and copies the following objects to the executable file:

- Objects that are directly referenced in scripts
- Objects that are referenced in painters

Objects that are directly referenced in scripts

Pocket PowerBuilder copies objects directly referenced in scripts to the executable file. For example:

- If a window script contains the following statement, *w_continue* is copied to the executable file:

```
Open(w_continue)
```

- If a menu item script refers to the global function *f_calc*, *f_calc* is copied to the executable file:

```
f_calc(EnteredValue)
```

- If a window uses a pop-up menu using the following statements, *m_new* is copied to the executable file:

```
m_new   mymenu
```

```
mymenu = create m_new
mymenu.m_file.PopMenu(PointerX(), PointerY())
```

Objects that are referenced in painters

Pocket PowerBuilder copies objects referenced in painters to the executable file. For example:

- If a menu is associated with a window in the Window painter, the menu is copied to the executable file.
- If a DataWindow object is associated with a DataWindow control in the Window painter, the DataWindow object is copied to the executable file.
- If a window contains a custom user object that includes another user object, both user objects are copied.
- If a resource is assigned in a painter, it is copied to the executable file. For example, when you place a Picture control in a window in the Window painter, the bitmap file you associate with it is copied.

Which objects are not copied to the executable file

When it creates the executable file, Pocket PowerBuilder can identify the associations you made in the painter, because those references are saved with the object's definition in the library. Pocket PowerBuilder also identifies direct references in scripts, because the compiler saves this information.

However, Pocket PowerBuilder cannot identify objects that are referenced dynamically through string variables. To do so, it would have to read through all the scripts and process all assignment statements to uncover all the referenced objects. The following examples show objects that are not copied to the executable file:

- If the DataWindow object *d_emp* is associated with a DataWindow control dynamically using the following statement, *d_emp* is not copied:

  ```
  dw_info.DataObject = "d_emp"
  ```

- The bitmap files assigned dynamically in the following script are not copied:

  ```
  IF Balance < 0 THEN
     p_logo.PictureName = "frown.bmp"
  ELSE
     p_logo.PictureName = "smile.bmp"
  END IF
  ```

- The reference to window *w_go* in a string variable in the following window script is not found by Pocket PowerBuilder when building the executable file, so *w_go* is not copied to the executable file:

  ```
  window   mywin
  string   winname = "w_go"
  Open(mywin,winname)
  ```

Which objects are not copied to the dynamic libraries

When it builds a dynamic library, Pocket PowerBuilder does not inspect the objects; it simply copies the compiled form of the objects. Therefore, the DataWindow objects and resources (such as GIF files) used by any of the objects in the library--either specified in a painter or assigned dynamically in a script--are not copied into the dynamic library.

For example, suppose *test_dw.pkl* contains DataWindow objects and *test_w.pkl* contains window objects that reference them, either statically or dynamically. If you build a dynamic library from *test_w.pkl*, you must either include the DataWindow objects in a Pocket PowerBuilder resource file that is referenced by *test_w.pkl*, or build a dynamic library from *test_dw.pkl*, as described in "How to include the objects that were not found".

How to include the objects that were not found

If you use only the types of references that are included in the EXE or PKD files built by Pocket PowerBuilder (as described in "Which objects are copied to the executable file"), you do not need to do anything else to ensure that all objects get distributed: they are all built into the executable file or its dynamic libraries.

If you use the types of references described in the two previous sections ("Which objects are not copied to the executable file" and "Which objects are not copied to the dynamic libraries"), you must include the objects that were not found, using the methods that follow.

Distributing graphic objects

For graphic objects such as GIF and BMP files, you have two choices:

- Distribute them separately
- Include them in a PKR file, then build an executable file or dynamic Pocket PowerBuilder library that uses the resource file

Distributing DataWindow objects

For DataWindow objects, you have two choices:

- Include them in a PKR, then build an executable file or dynamic Pocket PowerBuilder library that uses the resource file
- Build and distribute a dynamic library from the PKL that contains the DataWindow objects

Distributing other objects

All other objects (such as windows referenced only in string variables) must be included directly in a dynamic library.

Table 22-2 summarizes resource distribution possibilities.

Table 22-2: Summary: options for distributing resources

Distribution method	Graphic objects	DataWindow objects	Other objects
As a separate file	Yes	No	No
In an executable or dynamic library that references a PKR	Yes	Yes	No
Directly in a dynamic library	No	Yes	Yes

Listing the objects in a project

After you have built your project, you can display a list of the objects in the project from the Project painter, with three columns that show:

- The source library that contains the object
- The name of the object
- The type of the object

The report lists the objects that Pocket PowerBuilder placed in the executable file and the dynamic libraries it created when you built the project.

You can resize and reorder columns in the report just as in grid DataWindow objects. You can also sort the rows and print the report using the Sort and Print buttons.

To list the objects in a project:

1. **Build your project.**
2. **Select Design>List Objects from the Project painter menu bar.**

Troubleshooting errors during CAB file generation

After you build a project with the Build CAB File for Distribution option, you can check the Pocket PowerBuilder Output window or the *Err.log* file for a description of any errors in the build process.

If something goes wrong in the creation of the INF file or the BAT file, or in executing the *cabwiz.exe* to create the final CAB files, you are likely to see the following message in the Output window:

```
CAB Information File Generation Error
```

If no error messages display in the Output window, but a CAB file is still not generated, the error might be do to one of the following reasons:

- The generated INF file is somehow incorrect
- The generated BAT file is somehow incorrect
- The *cabwiz.exe* or *makecab.exe* are corrupted or missing from the Pocket PowerBuilder *support\cabwiz* directory

You can check the INF and BAT files in any text editor and you can reinstall the *support\cabwiz* directory from the Pocket PowerBuilder setup program. If you still cannot generate a CAB file from the Pocket PowerBuilder project, you should clear the Build CAB File for Distribution check box in the Project painter.

The Output window displays the following message when CAB file generation is successful:

```
CAB Information File Generation Starting

Created:  D:\DirectoryName\ExeName.inf

Created:  D:\DirectoryName\ExeName_makecab.bat

CAB File(s) Generated in directory: D:\DirectoryName

------- Finished Deploy of ExeName_test
```

In this output message example, *DirectoryName* is the name of the project directory and *ExeName* is the name of the project executable.

Appendixes

Appendix A describes the extended attribute system tables and Appendix B describes differences between the PowerBuilder and Pocket PowerBuilder products.

CiteSeer Find: powerBuilder and DDE Documents Citations

Searching for **powerbuilder and pocket**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
No documents match Boolean query. Trying non-Boolean relevance query.
876 documents found. **Only retrieving 250 documents (System busy - maximum reduced).** Retrieving documents...
Order: relevance to query.

Specification-based Prototyping for Embedded Systems - Thompson, Heimdahl, Miller (1999) (Correct) (5 citations)
successful. Languages in these domains, such as **PowerBuilder** and Visual Basic, provide powerful, high-level
www.cs.umn.edu/crisys/publications/thompson-fse99.ps

Evaluating Spatial and Textual Style of Displays - Ben Shneiderman (1995) (Correct)
as Visual Basic (Microsoft Corp.Reality, or **PowerBuilder**, and more complex cross-platform systems such
ftp.cs.umd.edu/pub/papers/papers/ncstrl.umcp/CS-TR-3451/CS-TR-3451.ps.Z

A New Lower Bound for Kernel Searching - Anderson, Lopez-Ortiz (Correct)
The connected components of P n V P (p) are called **pockets**. The boundary of a **pocket** is made of some
P n V P (p) are called **pockets**. The boundary of a **pocket** is made of some polygon edges and one line
of P only in its end points is called a chord. A **pocket** is said to be a left **pocket** if it lies locally to
www.cs.unb.ca/conf/cccg/eProceedings/37.ps.gz

Degree relatives are ordinary relatives - Butler (2001) (Correct)
1 (1) Peter ate everything that would fit in his **pocket**. Restrictive reading: Peter ate everything
A for Peter ate.Exhaustive reading: Peter ate a **pocket**full of something. 2 x(P(x) y(x y P(y)
(2) a. Peter ate everything which would fit in his **pocket**. b. Peter ate something that would fit in his
www-users.york.ac.uk/~lang21/ajb/papers/salt11-paper.pdf

Survey of Feature Research - Han (1996) (Correct)
removal, the feature set includes holes, slots and **pockets**, as depicted in Figure 1. The part shown in (a)
to be removed as machining features hole slot **pocket** Figure 1: feature examples linear sweep height
y z cutter sweep cutter and sweep path direction **pocket =pocket** profile sweep (c) **pocket** definition
graphics.skku.ac.kr/paper/IRIS-96-346.ps

Pocket Pavilion: A Synchronous Collaborative Browsing.. - Philip Mckinley And (1999) (Correct)
Pocket Pavilion: A Synchronous Collaborative Browsing
cse.msu.edu ABSTRACT This paper describes **Pocket** Pavilion, a web-based application that extends
handheld computing platforms running Windows CE. **Pocket** Pavilion is built atop Pavilion, an
ftp.cse.msu.edu/pub/crg/PAPERS/icme00-pocket.pdf

Paying for Health Care: Quantifying Fairness.. - Adam Wagstaff.. (Correct)
the arguments and methods with data on out-of-**pocket** payments from Vietnam in 1993 and 1998. This is a
of health spending in that country was paid out-of-**pocket** in 1998. We find that out-of-**pocket** payments had
paid out-of-**pocket** in 1998. We find that out-of-**pocket** payments had a smaller disequalizing effect on
econ.worldbank.org/files/2601_wps2715.pdf

Side-Chain Flexibility in Proteins Upon Ligand Binding - Rafael Najmanovich Josef (2000) (Correct)
to small side-chain rearrangements in the binding **pocket** residues. The analysis of side chain flexibility
database. The number and identity of binding **pocket** residues that undergo side-chain conformational
in general, only a small number of residues in the **pocket** undergo such changes (e.g.85% of cases show
www.weizmann.ac.il/home/ferafael/Najmanovich2000.pdf

The Itsy Pocket Computer - Bartlett, Brakmo, Farkas, Hamburgen, .. (2000) (Correct) (1 citation)
B E R 2 0 0 0 WRL Research Report 2000/6 The Itsy **Pocket** Computer Joel F. Bartlett Lawrence S. Brakmo
mail it to: WRL-Techreports@pa.dec.com The Itsy **Pocket** Computer Joel F. Bartlett, Lawrence S. Brakmo,
y Deborah A. Wallach Abstract The Itsy **pocket** computer is a powerful information device small
ftp.digital.com/pub/Digital/WRL/research-reports/WRL-TR-2000.6.ps.gz

Real-Time Restoration of Lens Distorted Images by Digital.. - Boer Nijmeijer (Correct)
be corrected by the system: barrel, pincushion and **pocket**-handkerchief. Applications can be found in very
with the distance to the optical axis. Barrel **Pocket**-handkerchief Pincushion Figure 1: Three kinds of
rotation symetric, and produces images like the **pocket**-handkerchief in Figure 1. To be able to correct
utelnt.el.utwente.nl/publications/../research/vlsi/ProRisc94.ps.gz

Investigating Touchscreen Typing: The effect of.. - Sears, Revis.. (1993) (Correct) (4 citations)
is not practical. Applications include portable **pocket**-sized or palmtop computers, messaging systems,
in even more situations, such as palmtop and **pocket**-sized computers, portable message systems, and
alternative. Touchscreens can be used with **pocket**-sized or palmtop computers, something that is
ftp.cs.umd.edu/pub/papers/papers/ncstrl.umcp/CS-TR-2662/CS-TR-2662.ps.Z

Formal Specification of Catalysis Frameworks - Filipe, Lau, Ornaghi, Taguchi.. (2000) (Correct)
a person plays the Company PersonAsEmployee Person **pocket**: Money receive_pay(amt: Money) pre: post:
Money) pre: post: work(has worked before **pocket** increased by amt pre: **pocket**500 worksfor Figure
has worked before **pocket** increased by amt pre: **pocket500** worksfor Figure 1. PersonAsEmployee
www.cs.man.ac.uk/~kung-kiu/pub/apsec00.ps.gz

Power and Energy Characterization of the Itsy Pocket.. - Flinn, Farkas, Anderson (2000) (Correct) (1 citation)
Power and Energy Characterization of the Itsy **Pocket** Computer (Version 1.5) Jason Flinn Keith I.
Power and Energy Characterization of the Itsy **Pocket** Computer (Version 1.5) Jason Flinn Keith I.
of the power consumption of the Itsy **Pocket** Computer Version 1.5 [1]a state-of-the art
ftp.digital.com/pub/Digital/WRL/research-reports/WRL-TN-56.ps.gz

The Itsy Pocket Computer Version 1.5 User's Manual - Viredaz (1998) (Correct)
J U L Y 1 9 9 8 WRL Technical Note TN-54 The Itsy **Pocket** Computer Version 1.5 User's Manual Marc A.
The Itsy **Pocket** Computer Version 1.5 User's Manual Marc A.
Marc A. Viredaz July 1998 Abstract The Itsy **pocket** computer is a flexible research platform
ftp.digital.com/pub/Digital/WRL/research-reports/WRL-TN-54.ps.gz

Flipturning Polygons (Extended Abstract) - Aichholzer, Cortés.. (Correct)
Godfried T Toussaint [Figure 1. A flipturn. The **pocket** is bold (red)and its lid is dashed. A central
1973 paper of Joss and Shannon [4] as follows. A **pocket** of a nonconvex polygon P is a maximal connected
The line segment joining the endpoints of a **pocket** is called the lid. A flipturn rotates a **pocket**
cgm.cs.mcgill.ca/~godfried/publications/flipturn.abstract.ps.gz

Constraint Optimization and Feature-Based Model.. - Germain, Stark.. (1996) (Correct) (1 citation)
outline the processes involved in fitting profile **pockets**, the most complex feature currently handled by
handled by the system, to sensed data. A profile **pocket** is a 2 1 2 D feature consisting of a
3-D points corresponding to a particular profile **pocket** starts with the user sketching the profile
www.cs.utah.edu/projects/robot/sam/../papers/Sam_Papers/constraint.pdf

Comparative body size relationships in pocket gophers.. - Morand, Hafner, Page.. (2000) (Correct)
On Comparative Body Size Relationships In **Pocket** Gophers And Their Chewing Lice Serge Morand 1
contrasts to study body size relationships between **pocket** gophers and their chewing lice, a host-parasite
between body size and hair-shaft diameter in **pocket** gophers, and that there is also a positive
taxonomy.zoology.gla.ac.uk/rod/papers/moran.pdf

TRANSIMS Data Preparation Guide - Bush, Berkbigler, al. (1998) (Correct)
5. **Pocket** Lane
14 4. **Pocket** Lane Table
is considered permanent if it is not a temporary, **pocket** lane (see the definition of **pocket** lane below)A
transims.tsasa.lanl.gov/PDF_Files/LAUR98-1411.pdf

The Erdös-Nagy Theorem and its Ramifications - Toussaint (1999) (Correct) (3 citations)
the convex hull of the polygon. If there are no **pockets** do not perform a flip. If there are **pockets** then
are no **pockets** do not perform a flip. If there are **pockets** then reflect one **pocket** across its line of
a flip. If there are **pockets** then reflect one **pocket** across its line of support of the polygon to
www-cgrl.cs.mcgill.ca/~godfried/publications/erdos.ps.gz

Documents 21 to 40 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP each article to make it easy to send out the source code without having to be concerned about sending a database with the sample code.

The actual HTML generation is accomplished with a PowerScript Describe function call. A small component can be build by creating a DataStore, retrieving data, and then calling the Describe function to return the HTML. The code is shown below in Code Sample #1. Notice that this code does all the necessary functionality in one function and that the function takes no arguments and returns a string.

Code Sample #1

```
datastore        lds_department
string           ls_html

//
// connect to database
//
SQLCA.DBMS="odbc"
SQLCA.dbparm="connectstring='dsn=eas demo db v3'"
CONNECT Using SQLCA ;
if SQLCA.SQLCode < 0 then return ""

lds_department = CREATE datastore
lds_department.dataobject = "d_department"
lds_department.SetTransObject(SQLCA)
lds_department.Retrieve()

ls_html = lds_department.Describe("DataWindow.Data.HTML")

return ls_html
```

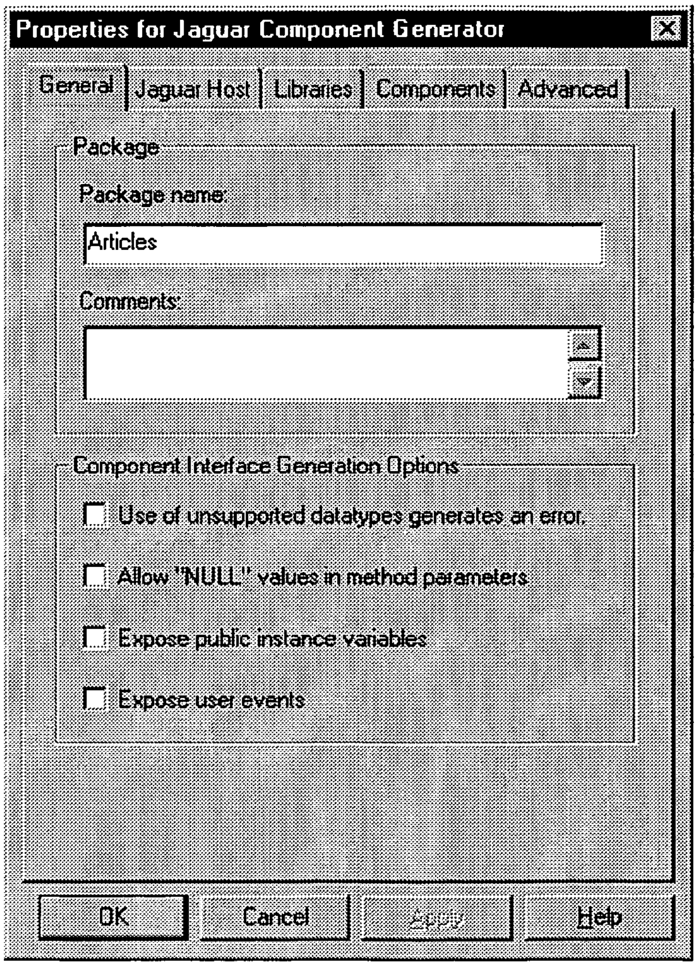
Deploying To Jaguar

Once the component is built, it needs to be deployed to Jaguar. This is done by using the PowerBuilder Project Painter and building a Jaguar component. What this does is create a CORBA component and install the package and component into Jaguar. Since there are articles available on the SDN that demonstrate how to create a Jaguar component, I will not review each screen, but only summarize the settings for this component. The jaguar connection information settings the defaults when it is initially installed and running locally on your machine.

Property	**Setting**
Package Name	Articles
Jaguar Host	Localhost
Port Number	9000
Login ID	Jagadmin
Include Unreferenced Objects InConsolidated PBD	Checked
Consolidate New Libraries By Default	Checked
Component Name	WebDW_Component
Component Type	Standard

Transaction Support	Requires Transaction
Support Instance Pooling	Check this before depolying into production.

Screen #1, Jaguar Component Generator General Tab


Properties for Jaguar Component Generator

General | Jaguar Host | Libraries | Components | Advanced

Package

Package name:

Articles

Comments:

Component Interface Generation Options

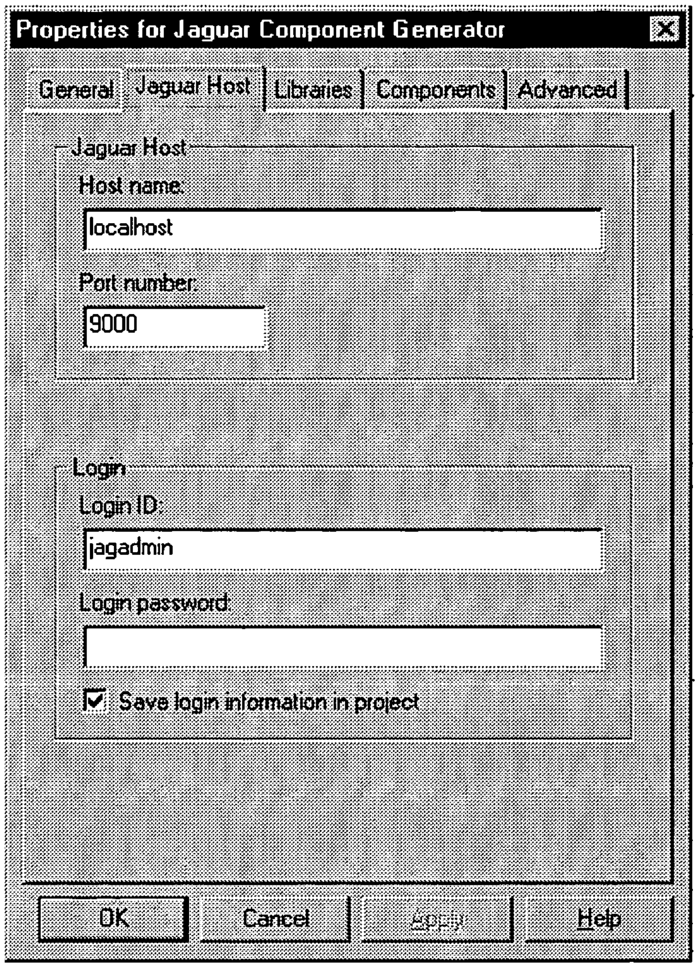
☐ Use of unsupported datatypes generates an error.

☐ Allow "NULL" values in method parameters

☐ Expose public instance variables

☐ Expose user events

Screen #2, Jaguar Component Generator, Jaguar Host Tab


Properties for Jaguar Component Generator

Jaguar Host

Host name: localhost

Port number: 9000

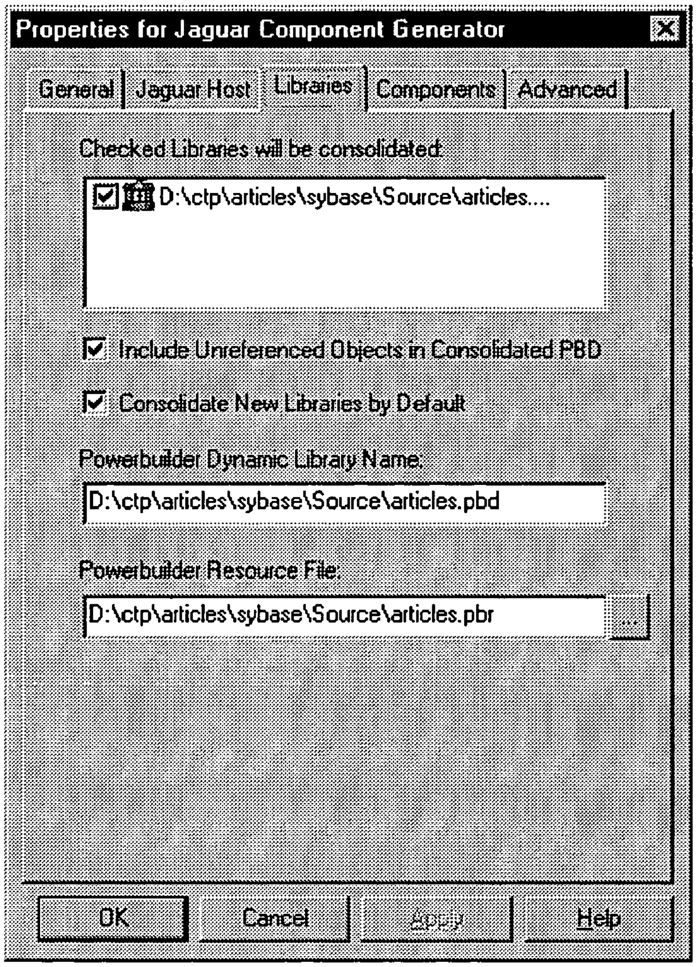
Login

Login ID: jagadmin

Login password:

[✓] Save login information in project

Screen #3, Jaguar Component Generator, Libraries Tab



Properties for Jaguar Component Generator

General | Jaguar Host | Libraries | Components | Advanced

Checked Libraries will be consolidated:

[✓] D:\ctp\articles\sybase\Source\articles....

[✓] Include Unreferenced Objects in Consolidated PBD

[✓] Consolidate New Libraries by Default

Powerbuilder Dynamic Library Name:

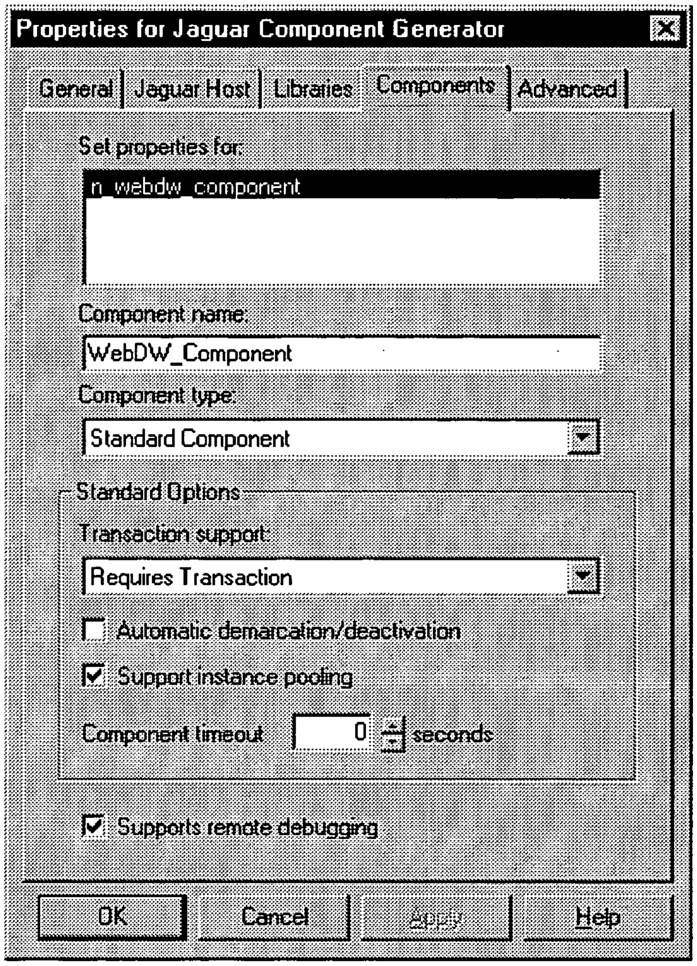
D:\ctp\articles\sybase\Source\articles.pbd

Powerbuilder Resource File:

D:\ctp\articles\sybase\Source\articles.pbr

OK | Cancel | Apply | Help

Screen #4, Jaguar Component Generator, Components Tab


Properties for Jaguar Component Generator

Set properties for:
n_webdw_component

Component name:
WebDW_Component

Component type:
Standard Component

Standard Options

Transaction support:
Requires Transaction

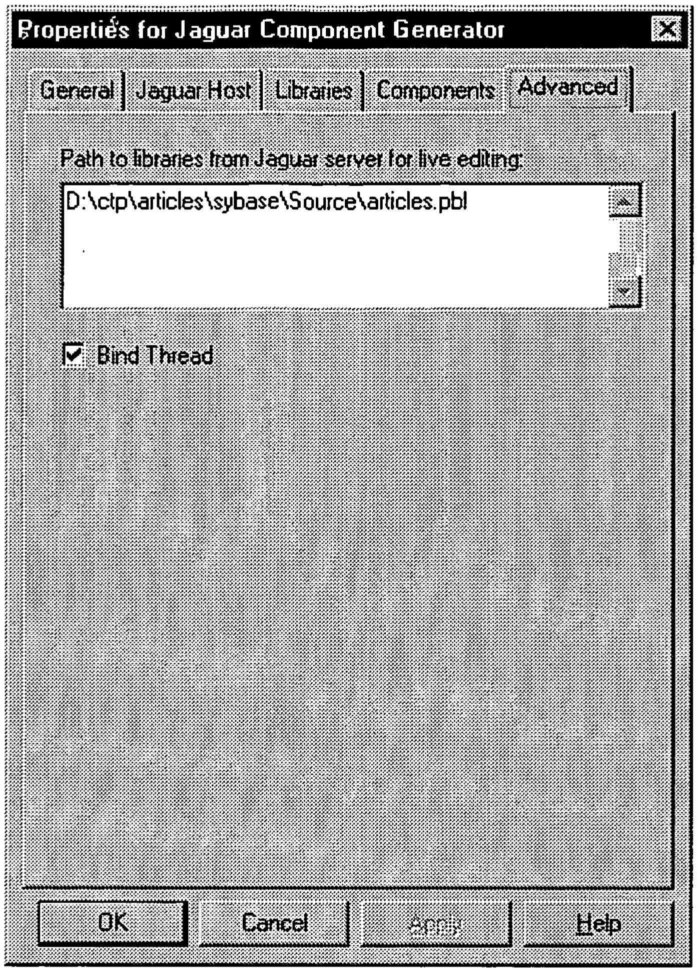
☐ Automatic demarcation/deactivation

☑ Support instance pooling

Component timeout 0 seconds

☑ Supports remote debugging

Screen #5, Jaguar Component Generator, Advanced Tab



Path to libraries from Jaguar server for live editing:

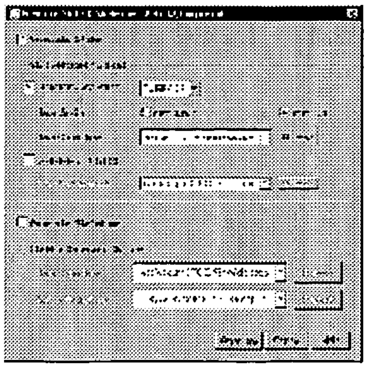
D:\ctp\articles\sybase\Source\articles.pbl

☑ Bind Thread

Generating Java Stubs And Compiling

Since this component will not be accessed from a PowerBuilder application, some type of stub needs to be generated and compiled to access the component methods. Using Java stubs is the manner that I have been creating Web DataWindow components since the EAStudio beta, and is what we will use for this article. The Java stubs are compiled by right clicking on the component name in the Jaguar manager and selecting Generate Stubs/Skeletons. A modal window will appear and you will check Generate Stubs and Generate Java Stubs. Leave all other options with the default values. Screen #6 shows the stub generation dialog window.

Screen #6, Generate Stubs



Once the Java stubs have been generated, they must be compiled. These are Java programs that provide access to the component methods. These stubs are placed into the %jaguar%\html\classes\packagename directory. Since I named the package Articles, the directory on my computer is d:\Sybase\Jaguar CTS 3.5 \html\classes\Articles. The Java compiler is named javac.exe, and the command line to compile the stubs is javac *.java. A successful compilation is indicated with the command prompt returning without any messages. If you get any messages, check to see that the %jaguar%\html\classes directory is included in the system classpath variable.

JavaScript

There is very little JavaScript necessary to access the component. The script simply needs to create an instance of the component and call the method to generate the HTML. The code is shown below. Keep in mind that this is server-side script that the user will never see, they will only see the generated HTML.

```
lds_department.Modify
("DataWindow.htmlgen.
SelfLink='" + s_selflink_nar
lds_department.Modify
("DataWindow.htmlgen.
SelfLinkArgs='"
+ s_selflink_args + "'
"
```

Determining Browser Type

You will notice that the generated HTML is not all that appealing. This is because the component did not know the type of browser was being used. If PowerBuilder knows the browser type, then the generated HTML will be optimized for the browser and version. This is accomplished by adding a Modify function call to set the DataWindow.htmlgen.browser property of the DataStore. The browser type must be pass in from the client machine because the HTTP request has information about the client browser. We will add an argument of type string to our method that indicates the browser type (s_browser). Then a PowerScript Modify function will set the attribute as shown below.

```
    lds_department.Modify("DataWindow.htmlgen.Browser='" + s_browser + "' ")
```

The server-side script that calls the component method must now pass the browser type. This is done by the PowerDynamo GetServervariable function. The following code places the browser type into a script variable, lv_browser, that is passed into the of_dw1 component method. I recommend taking a look what the lv_browser variable contains as it can be used when implementing support for browser types (i.e. Internet Explorer, Netscape). This discussion will occur in a future article.

```
    lv_browser = document.GetServerVariable("HTTP_USER_AGENT");
document.write(dwServer.of_dw(lv_browser));
```

HTML Object Name

Now that the page looks acceptable, it is necessary to provide a reference name to the generated object that can be used for client-side scripting. This is done in the HTMLGenerator component by calling SetHTMLObjectName and passing the name to assign as the argument. This is another thing that is accomplished with a PowerScript Modify function call to the DataWindow.htmlgen.ObjectName property of the DataStore. Our component will add the line of code into the method so that an argument is not necessary. The object name will be dwArticle, **and please not that this is case sensitive.** The code is shown below.

```
    lds_department.Modify("DataWindow.htmlgen.ObjectName='dwArticle'")
```

The result of this being added to the method is not immediately obvious. However, if you looked at the errors generated by the previous screen in the browser, there was an invalid reference to an HTML object name. This error goes away once this is added to the component.

SelfLink

The purpose of the SelfLink is to maintain any passed page parameters when the page is reloaded. A reload occurs upon any action that requires a round trip to the server such as inserting a new row or performing an update. This property can be set in the DataWindow painter or in the component. We will code this in our component and add 2 new argument to the method, selflinkname and selflink_args, both of type string. These attributes of the DataStore are once again set with Modify function calls and the code is shown below.

```
lds_department.Modify
("DataWindow.htmlgen.
SelfLink='" + s_selflink_nar
lds_department.Modify
("DataWindow.htmlgen.
SelfLinkArgs='"
+ s_selflink_args + "'
"
```

Dropdown DataWindow With Retrieval Arguments

The premise of building a customized Web DataWindow component is to provide functionality that is not available in the HTMLGenerator component. Populating a child DataWindow that uses retrieval arguments is a common requirement that is not fulfilled with the HTMLGenerator component. Let's add more functionality to our component to handle this.

The main DataWindow, d_department, contains a dropdown for Manager ID, d_dddw_sales_reps. This DataWindow is included in the example application installed with PowerBuilder. Since it had no retrieval arguments, I went an added an argument for employee_id that is retrieves all records with an employee ID greater than or equal to the argument.

The method on our component will be modified to have 2 more arguments, column name and retrieval argument value. If this information is passed to the component, the following things will occur.

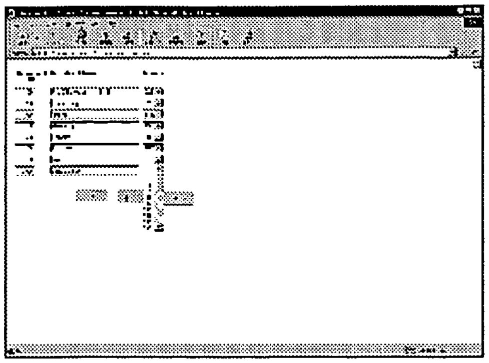
- Get the handle to the child DataWindow with the GetChild function
- Set the transaction object for the child DataWindow
- Retrieve the child DataWindow with the passed argument

If this is not done, the PowerBuilder "Specify Retrieval Arguments" modal window will appear. The code for this functionality is shown below. Naturally, this is a simple example and a production method will have more robust functionality to handle all dropdown DataWindows on a page.

```
     if (s_column <> "") then
    lds_department.GetChild(s_column,ldwc)
    ldwc.SetTransObject(SQLCA)
ll_temp = Long(s_column_arg)
    ldwc.Retrieve(ll_temp)
end if
```

Screen #7 shows the screen with the dropdown limited to employees with an id of 1000 or greater. This was accomplished by passing the column name and "1000" as the retrieval argument for the dropdown DataWindow.

Screen #7, Dropdown Limited



Entire JavaScript Code

This article broke down several steps in the building of the component. Following is the complete JavaScript code script that generated our Web DataWindow.

```
lds_department.Modify
("DataWindow.htmlgen.
SelfLink='" + s_selflink_nar
lds_department.Modify
("DataWindow.htmlgen.
SelfLinkArgs='"
+ s_selflink_args + "'
"
```

Summary

This article demonstrated how simple it is to build a Web DataWindow component. The reason I recommend this approach is that it give you, the developer, complete control over how the component functions. It can be customized to fit your company needs for all your Web applications.

If you would like a copy of this code along with the DataWindow objects, feel free to email me and I will make it available. Because this example uses the Enterprise Application Server Demo database, it is easy to make this example work on any machine. In the future, I will make the code from every article available on our web site for download.

If you have any ideas on future articles about the Web DataWindow, please forward an email and I will do my best to write about it. If you have a question about a technique, please contact me about it. Chances are that there are others who have the same question. In addition, please pass along any comments you have on this new and exciting feature of EAStudio.

About the Author

Larry Cermak is Vice President of Technology with Corporate Technology Partners, Inc., an emerging technology consulting and training firm specializing in enterprise solutions. Check out their web site at www.ctpartner.com about consulting and training services. Larry is a member of Team Sybase, a CPD Professional, a member of the CPD Review Committee, a writer for the Sybase Developer Network (SDN) and PowerTimes journal, and a frequent speaker at Powersoft/Sybase conferences and seminars across the country. Larry can be reached at 630-428-2650 or at lcermak@ctpartners.com.

Back to Top

Related Links

- Building Applications Using PowerBuilder and EAServer (EAD310) - 3 Days

- How to install Powerbuilder 7.0 self-study on a PC

Document Attributes

Content Id: 1003254	**Last Revised:** March 07, 2000
Part Number: N/A	**Authorization:** Public
Focus: Technical	**Career:** Developer/DBA
Career Interest: Developer/SDN	**Document Library:** White Paper-Technical
Hardware Platform: Windows	**Products:** PowerBuilder
Solutions: Internet Applications	**Expertise Level:** 4-all
Technical Topics: Application Development , Internet Application Development	

Web DataWindows as Criteria Screens

This article, number six in Larry Cermak's series on the Web DataWindow, explores how a Web DataWindow can be used as a criteria screen for a report or a query.

By: Larry Cermak,
Vice President of Technology,
Corporate Technology Partners, Inc.

Click here to download the pdf version of this document.

Welcome to article number six in my series on the Web DataWindow®. Previous articles discussed basic configuration, master detail processing, client-side scripting, and server-side scripting. In this article, we take a look at a commonly asked question: "How can a Web DataWindow be used as a certain criteria screen for a report or query?" Below, I use an example to illustrate how you can accomplish this using an external DataWindow.

Article Outline

Design

The example used here is a customer query screen that can search for a customer by ID number, state, or zip code. This criteria screen needs to include three fields so that end users can enter the appropriate information, and it must include a search button that, when pressed, starts the search. Additionally, the design calls for the search criteria to be on the same sreen as the results, which requires the use of and an HTML frame.

Back to article outline.

HTML Frame Setup

The easiest way to create an HTML frame set is to use the PowerSite™ development environment to make the screen look the way you want it to appear. PowerSite generates the HTML code automatically.

Because there are only three feilds on the search criteria screen, let's split the screen vertically, one field per section. The left 30% of the screen contains the criteria screen, s_criteria.html, and the right 70% of the screen contains the results screen, s_results.html.

Below is the code that PowerSite generates for our frame set, which is named f_search.html. Notice that the HTML pages for the source and results have not been created yet. Opening the frame at this point will cause an error because the page will not be found, but we can leave the code this way for now.

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET COLS="30%,*">
   <FRAME NAME=left SRC="s_criteria.html">
   <FRAME NAME=right SRC="s_results.html">
</FRAMESET>
</HTML>
```

Click here to download code.

Back to article outline.

Criteria DataWindow

Next we need to build a way for end users to enter search criteria. An external DataWindow used as a Web DataWindow is the answer here.

However, for an external DataWindow to work as well as a Web DataWindow and a criteria screen, two features need to be in place. First, for the DataWindow to allow fields to be entered in a browser, it must have update characteristics. You can enter a dummy name as the table name. PowerBuilder® will inform you that the table does not exist. That is OK for now because we will not be trying to update the database.

Second, a user-defined button should be used to initiate the search. PowerSite generates HTML for links based on the field values that exist when the DataWindow is retreived. Because the fields on a criteria screen have not been entered at the point of generation, a link argument will have null values. A user-defined button addresses this by using client-side scripting to read the entered values with the GetItem function.

We need a client-side script to detect the button click and pass the arguments. Remember to select the Client Events and Client Scriptable options on the HTML Generation tab on the DataWindow, as shown in Figure 1. We explore the client-side script a bit further down.

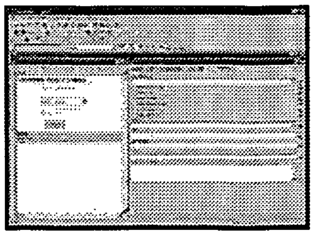


Figure 1: HTML generation options for the criteria DataWindow.

Back to article outline.

Results DataWindow

The results DataWindow can be any format the design specifies. Our example uses a tabular format. Notice that none of the options on the HTML Generation tab are checked—this DataWindow is view only. The DataWindow can have retrieval arguments, or a "where" clause can be added at the time the data is retrieved. For this example, we add the "where" clause with method calls to the component.

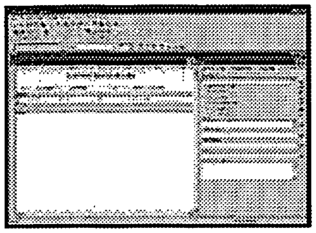


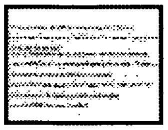
Figure2: Results DataWindow.

Back to article outline.

Server-side Scripting for the Criteria Screen

The JavaScript code that generates the criteria screen is the same code that has generated all the DataWindows in my article. As a fresher, let's review the main lines of code. They are available below.

Each main line is shown below along with a comment that describes what the line does. Naturally, a production script should include error checking to make sure method calls are successful and to take appropriate action if necessary.



Click here to download code.

Notice there is no Retrieve method in this code to retrieve the data. The reason is that if a Retrieve method is issued against an external DataWindow or DataStore, an error occurs.

There is also no InsertRow method called to insert a row into the DataWindow. This method does not exist on the HTML Generator component.

How, then, do we insert an empty row into our DataWindow? The window.onload event can be used to call the InsertRow function to insert a new row into the DataWindow, as shown below.

```
<BODY language=JavaScript onload="dwMine.InsertRow(0);">
```

Click here to download Code

This statement will, however, cause a loop when the window is loaded. A new row is inserted when the window is loaded, which causes the page to be reloaded. The page is reloaded, and the window.onload event fires again, which inserts another row and causes the page to load again.

This cycle can be broken if you check that there is already a row in the DataWindow before doing the insert. As with a PowerBuilder application, this checking is accomplished with the RowCount function:

```
<BODY language=JavaScript onload="if (dwMine.RowCount() == 0){dwMine.InsertRow(0);}">
```

Click here to download Code

Back to article outline.

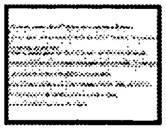
Client-side Scripting for the Criteria Screen

The criteria window will detect when the search button is pressed, load the results screen, and pass the three criteria fields as arguments—ID, state, and zip code. Let's go over what happens in the example script below.

First, a client-side script like the example below traps the ButtonClicked event in the browser and checks whether the button clicked is named b_search. When this button is clicked, the script receives the values for the three fields with GetItem function calls.

Next, the link command is created. It consists of the URL of the window to open and the parameters to pass. Parameters are identified by a question mark, ?, followed by the name and value. An ampersand, &, rather than a question mark indicates subsequent parameters.

Finally, the syntax to load the results window is specified and loaded into the HTML frame target. The keyword *parent* identifies the frame, *right* is the frame name assigned by the frame set, and *location.href* is the URL to load.

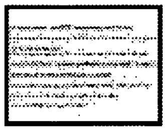


Click here to download code.

Back to article outline.

Server-side Scripting for the Results Screen

For the results screeen, the JavaScript code is similar to taht of the criteria screen. The only difference is that the code for the results screen will add a "where" clause to the SQL statement and then retrieve the data. The script checks the passed parameters, in the same manner as PowerBuilder checks passed parameters, to determine how to construct the "where" clause. Each parameter that is not equal to "null" will add additional elements to the "where" clause in the code shown below.



Click here to download code.

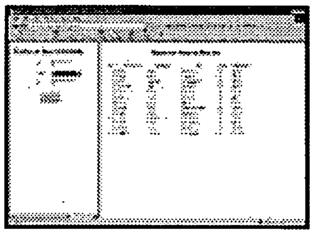


Figure 3: Finished product.

Back to article outline.

Summary

Using an external DataWindow to enter search or report criteria is a common technique in PowerBuilder applications. This article demonstrated how easily this functionality can be moved to the Web using existing DataWindows and a little bit of JavaScript coding to generate a Web DataWindow.

If you would like a copy of this code along with the DataWindow objects, feel free to email me and I will make it available. Because this example uses the Enterprise Application Server™ Demo database, it is easy to make this example work on any machine.

If you have any ideas about future articles about the Web DataWindow, please forward an email, and I will do my best to write about it. If you have a question about a technique, please contact me. Chances are that there are others who have the same question. In addition, please pass along any comments you have on this new and exciting feature of Enterprise Application Studio™.

Back to article outline.

About the Author

Larry Cermak is the Vice President of Technology with Corporate Technology Partners, Inc., an emerging technology consulting and training firm specializing in enterprise solutions. Larry is a member of Team Sybase, a CPD Professional, a member of the CPD Review Committee, a writer for the Sybase Developers Network (SDN) and *PowerTimes* journal, and a frequent speaker at Powersoft/Sybase conferences and seminars across the country. Larry can be reached at 630-428-2650 or at lcermak@ctpartners.com.

Back to Top

Document Attributes

Content Id: 1002379	**Last Revised:** November 19, 1999
Authorization: Public	**Focus:** Technical
Career: Developer/DBA	**Career Interest:** Developer/SDN
Document Library: White Paper-Technical	**Hardware Platform:** Windows
Products: PowerBuilder	**Solutions:** Internet Applications
Expertise Level: 4-all	**Technical Topics:** DataWindows , Application Development , Internet Application Development

Deployment Files for PowerBuilder, InfoMaker, and the HTML DataWindow

This document provides a list of files you need to deploy with applications or components you build in PowerBuilder or InfoMaker or that use the HTML DataWindow.

Deployment Files for PowerBuilder, InfoMaker, and the Web DataWindow

About this document
This document provides a list of files you need to deploy with applications or components you build in PowerBuilder 7 or InfoMaker 7 or that use the Web DataWindow.

HTML DataWindow
The Web DataWindow is also called the HTML DataWindow; both terms refer to the same technology.

More information about specific deployment scenarios is available in *Application Techniques*, the *InfoMaker User's Guide*, the *DataWindow Programmer's Guide*, and *Using the PowerBuilder Internet Tools.*

Contents

Using this document

Use this document to determine which files you need to deploy with Windows applications developed in PowerBuilder or InfoMaker or with the tools provided with PowerBuilder.

This document is intended to help you write installation programs using a third-party software package that creates installation configurations. It tells you which files each user needs, where you can find the files, where they should be installed, and what registry settings need to be made.

Two-tier applications

Conventional client/server applications, in which a PowerBuilder or InfoMaker client accesses a database server, need the following files on the client computer:

- The executable file and any supporting dynamic libraries and resources

- PowerBuilder runtime files
- Files needed to access the database

Multitier applications

Distributed and Web applications require different sets of files to be deployed on users' computers and servers. Use the following table to locate information about the specific files you need to deploy with your application. For most deployment scenarios, you will find more detailed information in the online or hardcopy documentation for each tool.

Scenario	See these sections
PowerBuilder or InfoMaker client application accessing data on a database server	"PowerBuilder runtime files" "Database connections"
PowerBuilder client that does not require database access (such as a client using a Jaguar server component for database access)	"PowerBuilder runtime files"
Distributed PowerBuilder server application accessing data on a database server	"PowerBuilder runtime files" "Database connections"
Jaguar component created in PowerBuilder	"PowerBuilder component on Jaguar server"
COM/MTS component created in PowerBuilder	"PowerBuilder COM servers"
Web application using Web DataWindow with PowerDynamo and Jaguar	"Files required on the Jaguar server" "Files required on the PowerDynamo server"
Web application using Web DataWindow with PowerDynamo and MTS	"Files required on the MTS or IIS server" "Files required on the PowerDynamo server"
Web application using Web DataWindow with ASP and MTS or IIS	"Files required on the MTS or IIS server" "Files required on the ASP server"
Web application using the DataWindow Web control for ActiveX	"DataWindow Web control for ActiveX"
Web application using PowerBuilder plugins or the window ActiveX	"Plugins and PowerBuilder window ActiveX controls"

Installed and deployment paths

The *Installed path* listed after some of the tables in this document is the location where files are installed when you install PowerBuilder or another Enterprise Application Studio tool and select the default installation location. The *Deployment path* tells you where those files can be copied to on the computer where you install your application or component.

App Path registry key

Some tables are followed by a list of the *Registry entries* your installation program needs to make so that your application or component can find the files it needs. When an application runs on Windows, it looks for supporting files in these locations and in this order:

1. The directory where the executable file is installed.
2. The Windows system and Windows directories (on NT, in C:\WINNT\SYSTEM32, C:\WINNT\SYSTEM, and C:\WINNT).
3. In an application path that can be specified in the registry.
4. In the system path.

You don't need to specify an application path, but it is recommended.

To specify the path the application uses to locate supporting files, your installation program should create an App Path key for your application in this registry location:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
                    CurrentVersion\App Paths
```

Set the data value of the (Default) string value to the directory where the application is installed and create a new string value called Path that specifies the location of shared files. The following example shows a typical registry entry for an application called MYAPP.EXE that uses Adaptive Server Anywhere. The registry key is enclosed in square brackets and is followed by string values for the key in the format `"Name"="Value"` :

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\
                    CurrentVersion\App Paths\myapp.exe]

"Default"="C:\Program Files\myapps\MYAPP.EXE"

"Path"="C:\Program Files\myapps;C:\Program Files

                    \sybase\shared\PowerBuilder;c:\program files

                    \sybase\Adaptive Server Anywhere 6.0\win32\;"
```

Note: *About REG files* Registry update files that have a .REG extension can be used to import information into the registry. The format used in registry key examples in this document is similar to the format used in registry update files but these examples are not intended to be used as update files. The pathnames in data value strings in registry update files typically use a pair of backslashes instead of a single backslash, and the "Default" string value is represented by the at sign (@). **About filenames**

Tables in this document show the ANSI versions of filenames. Some other versions of PowerBuilder have a letter at the end of the filename to distinguish among versions. When you deploy an application, deploy the same PowerBuilder DLLs with which you developed it.

PowerBuilder runtime files Note: *Database connectivity* Files required for database connectivity are listed separately in "Database connections".

The following table lists the core PowerBuilder runtime files. You only need to install the files that your application requires. For example, PBVM70 .DLL is required for all deployed applications, but you need to install PBRTC70.DLL only if your application uses Rich Text controls or RichText DataWindow objects.

Name	**Required for**
PBVM70.DLL	All
LIBJCC.DLL	All
PBDWE70 .DLL	DataWindows and DataStores
PBRTC70 .DLL	Rich Text

PBAEN70.TLB	OLE automation servers that use PowerBuilder.Application
PBFNT70.INI	Mapping unavailable fonts
PBLAB70.INI	Label DataWindow presentation style predefined formats
PBTRA70 .DLL	Database connection tracing

Installed path C:\Program Files\Sybase\Shared\PowerBuilder

Deployment path Same directory as the application, in a directory on the system path, or in the App Path registry key.

Registry entries See "App Path registry key".

Database connections

If you are deploying an executable or component that accesses a database, your users need access to the DBMS and to the database your application uses.

Note: *Where to install database connectivity files* You *do not need to deploy database connectivity files* with a client application that relies on a middle-tier component on another computer to perform database transactions. Database connectivity files must be deployed on the computer that interacts with the database server.

You need to:

- If necessary, install the DBMS runtime (client) files in the application directory or in a directory on the system path.
  If your application uses a standalone Adaptive Server Anywhere database, you can install the Adaptive Server Anywhere Runtime Edition files on the user's computer. For more information, see "Adaptive Server Anywhere files". Otherwise follow the instructions and licensing rules specified by the vendor.
- Make sure each user has access to the database the application uses.
  If your application uses a local database, install the database and any associated files, such as a log file, on the user's computer.

  If your application uses a server database, make sure the user's computer is set up to access the database. This may be the task of a database administrator.
- Install any database interfaces your application uses on the user's computer.
- If your application uses the ODBC interface, configure the ODBC database drivers and data sources, as described in "Configuring ODBC data sources and drivers".

For more information about database drivers, see:

- "Native database drivers"
- "ODBC database drivers and supporting files"
- "OLE DB database providers"
- "JDBC database interfaces"

Native database drivers

The following table lists the native database drivers supplied with PowerBuilder. If an application or component uses the database specified, the file is required on the computer. The first two characters of the native database filename are PB, the next three characters identify the database, and the last two identify the version of

PowerBuilder:

Name	Required for
PBIN770.DLL	INFORMIX I-Net 7
PBIN970.DLL	INFORMIX I-Net 9
PBMSS70.DLL	Microsoft SQL Server 6 and 7
PBO7370.DLL	Oracle 7.3
PBOR870.DLL	Oracle 8.0
PBO8470.DLL	Oracle 8.0.4 and later
PBDIR70.DLL	Sybase DirectConnect
PBSYC70.DLL	Sybase Adaptive Server Enterprise CT-LIB
PBSYJ70.DLL	Sybase Adaptive Server Enterprise CT-LIB for Jaguar deployment only

Installed path C:\Program Files\Sybase\Shared\PowerBuilder

Deployment path Same directory as the application, in a directory on the system path, or in the App Path registry key.

Registry entries See "App Path registry key".

Notes: When you deploy a PowerBuilder custom class user object to Jaguar, you need to use the SYJ database interface rather than SYC to connect to an Adaptive Server Enterprise database. You cannot use SYJ in the PowerBuilder development environment, but you can use the SYJ Database Profile Setup dialog box to set the appropriate connection parameters. You can then copy the syntax from the Preview tab into the script for your Transaction object.

ODBC database drivers and supporting files

This section lists files that are required for all ODBC database connections from PowerBuilder or InfoMaker applications, as well as files required for a specific database interface or DBMS.

PowerBuilder ODBC interface

The following PowerBuilder ODBC interface files are required if your application uses ODBC:

Name	Description
PBODB70.DLL	PowerBuilder ODBC interface
PBODB70.INI	PowerBuilder ODBC initialization file

Installed path C:\Program Files\Sybase\Shared\PowerBuilder

Deployment path Same directory as the application, in a directory on the system path, or in the App Path registry key.

Registry entries See "App Path registry key".

Notes: The INI and DLL files must be in the same directory. If you have modified the PBODB70 initialization file, make sure you deploy the modified version.

Microsoft ODBC files

The following Microsoft ODBC 3.5 files are required if your application uses ODBC:

Name	Description
DS16GT.DLL DS32GT.DLL ODBC32.DLL ODBC32GT.DLL ODBCAD32.EXE ODBCCP32.CPL ODBCCP32.DLL ODBCCR32.DLL ODBCINST.CNT ODBCINST.HLP ODBCINT.DLL ODBCTRAC.DLL	Microsoft ODBC driver manager, DLLs, and Help files

Installed path Windows system directory

Deployment path Windows system directory

Registry entries None

Notes: The Microsoft ODBC Driver Manager (ODBC32.DLL) and supporting files are usually already installed in the user's Windows system directory. You can use the MDAC_MIN.EXE setup file in the Support directory on Install Disk 2 to update users' systems if necessary.

Intersolv ODBC drivers and supporting files

The following Intersolv ODBC files are required if you use the database interface specified.

Note: *Optional Help files* Help (.HLP and .CNT) files need only be deployed if you expect users to perform database administration tasks.

Name	Required for
PBBAS13.DLL PBFLT13.DLL PBUTL13.DLL PBTRN13.DLL IVPB.LIC PBDRV13.CNT PBDRV13.HLP	All INTERSOLV drivers
PBBTR13.DLL PBBTR13.HLP	INTERSOLV Btrieve
PBDBF13.DLL PBDBF13.HLP	INTERSOLV dBASE

PBDB213.DLL PBDB213.HLP	INTERSOLV DB2
PBGUP13.DLL PBGUP13.HLP	INTERSOLV SQLBase
PBXLWB13.DLL PBXLWB13.HLP	INTERSOLV Excel Workbook
PBINF13.DLL PBINF13.HLP	INTERSOLV Informix
PBINF913.DLL PBINF913.HLP	INTERSOLV Informix 9
PBOING13.DLL PBOING13.HLP	INTERSOLV OpenIngres
PBOI13.DLL PBOI13.HLP	INTERSOLV OpenIngres 2
PBOR713.DLL PBOR713.HLP	INTERSOLV Oracle 7
PBOR813.DLL PBOR813.HLP	INTERSOLV Oracle 8
PBIDP13.DLL PBIDP13.HLP	INTERSOLV Paradox 5
PBPRO13.DLL PBPRO13.HLP	INTERSOLV PROGRESS
PBSS613.DLL PBSS613.HLP	INTERSOLV Microsoft SQL Server 6.0
PBSYB13.DLL PBSYB13.HLP	INTERSOLV Sybase SQL Server and Adaptive Server Enterprise
PBTXT13.DLL PBTXT13.HLP	INTERSOLV Text

Installed path C:\Program Files\Sybase\Shared\IntersolvODBC

Deployment path Same directory as the application, in a directory on the system path, or in the App Path registry key.

Registry entries See "App Path registry key" and "Configuring ODBC data sources and drivers".

Notes: The following PowerBuilder INTERSOLV drivers have been fully tested with PowerBuilder 7: Btrieve, DB2, dBASE, Excel Workbook, Paradox, and Text.

[Next to page 2]

Back to Top

Document Attributes

Content Id: 47953 **Last Revised:** August 19, 1999

Authorization: Public
Document Library: White Paper-Technical
Solutions: Not Solution Specific
Technical Topics: Application Development

Focus: Technical
Products: PowerBuilder
Expertise Level: 4-all

Building a System You Can Live with Later - Part 2

In part 2, we explore four additional areas of almost inevitable system change: batch processing, data duplication, security, and new technology.

By: Topher White
Internet Architect
National Web Technology Organization
AT&T Wireless Services

Introduction

In part 2, we explore four additional areas of almost inevitable system change: batch processing, data duplication, security, and new technology. For each of these areas, we'll look at how what those changes may be and how your initial application design can anticipate and handle them.

Article outline:

Batch Processing

Online, interactive computing promised the end of the batch-processing paradigm that dominated the early days of the industry. However, batch processing is not only alive and well, but batch demands are increasing in IT systems everywhere.

Batch processing is moving in time as well. What used to be restricted to the quiet night periods ("the batch window") now requires part of the day as well. And for many 24 x 7 systems, there is no such thing as a batch window to begin with.

The approach you select for constructing an application that supports batch jobs depends on system needs.

The concept to keep in mind when deciding how to build your application is the *aggregate effect*.

The aggregate effect of a batch job is the effect that it has on the entire system when run in context (that is, when run with all the other jobs and users who will be present during its execution cycle).

Batch jobs often have a great deal of difficulty interacting with one another. Additionally, batch processes often interfere with interactive users, and vice versa.

Original batch processing was exclusively batch (no interactive users) and exclusively serial (jobs were not done in parallel). Each job had the entire machine and data resources at its disposal. This is no longer the case. New jobs must then not be measured and tested in isolation, but be placed within the full environment *and the aggregate effect measured*.

Using Standard Transactions

The first method of building batch jobs that can coexist peacefully is to use standard transactions—transactions already defined and encapsulated within the online system. (The use of encapsulated transactions was discussed in the Upgrades section of part 1 of this article.) If the transactions are well defined and encapsulated on the server side, it doesn't matter whether they are driven by an online interactive user or by a batch job. In this case, the batch job looks like just another interactive user. The two transaction sources combine to form an additive input to a single modeling equation.

Individual batch job may be less efficient, but the benefits include:

- A better aggregate effect
- The ability for daytime jobs to coexist with interactive users
- Easier modeling of system performance as a whole; modeling the system behavior so that batch jobs behave differently at a fundamental level requires multiple equations and adds an order of magnitude to the task

Well-defined transactions encapsulated into a single server-side application are key to success. While use of standard transactions can be accomplished when the different programs (interactive client and batch job) use the same code for the transactions, it leads to problems. After several years, dozens of applications might share a block of transaction code. Upgrading these code blocks quickly becomes virtually impossible.

Feeds versus Extracts, and Using Secondary Data Stores

A second possible area of system inefficiency centers on a key use of batch jobs—moving data from one system to another. There are two ways to approach this: feeds and extracts. Feeds move data continuously or in very small increments and tend to include synchronization problems. An extract usually runs once a night and moves data in one large block but is prone to severe degradation as data grows because it is a linear data operation. You must balance the processes when moving data from one system to another.

One method of avoiding feed limitations without creating extract-like growth problems involves using a "secondary data store"—storing in a separate area a copy of data that has not been moved. You can use a secondary data store for extracts, reports, or any other job that generates output—even update jobs. (I introduced the concept of a secondary data store in Part 1 of this article in December's *Powerline*.]

A penalty exists for setting up a secondary data store. Each transaction must move data not just into the primary store, but also into the secondary. Because this can affect transaction performance, it may not be a viable option if you have a high transaction rate. However, it may be the only option if you are running a 24 x 7 operation.

The first use for data in a secondary data store is as the driver for selecting which rows to update in the primary stores. This assumes that the primary data update can be driven off of previous transactions.

You can also use the secondary data store for inserting, manipulating and transforming data. Then a single small transaction can update tables en masse.

For example, assume a report prints all new accounts. You could search the entire account list looking for accounts with a creation date equal to today's date. A faster search involves inserting the account number in the secondary data store at the time the account is created. The report then uses this number as driving data. Though creating of a new account requires a time investment because data must be put into two places, the aggregate affect is much less.

Managing Concurrency Issues

Another area of batch processing difficulty involves how the client application takes and holds locks. As I discuss in Section entitled "Growth" in Part 1 of this article), ensuring concurrency can be difficult as you grow. Ensuring concurrency can also create difficulties when you incorporate batch jobs. If an interactive client takes and holds a lock—for instance, an account lock while a representative is talking to a customer—a concurrent batch job must compete with that client.

The "Reserved by" field mentioned in Part 1 offers a solution. When you use this field and the batch job encounters that row, it can choose of two options.

- If the batch process is not conducting a competing operation (say, a report or a count), it can ignore the field and continue to process.

- If the batch process is conducting a competing operation (an update), it can move on to the next row and revisit this one later on in an exception process.

If you use a low-level lock, however, neither of these options is workable. The batch job must wait for the user to finish and then move on to the next row.

The "reserved by" field can also keep a batch job from blocking other jobs and user sessions . In this way, the online application does not sit and wait for the batch job to finish. It can respond with the same options listed above. Both the batch job and the interactive users spend their time processing instead of waiting.

Data Duplication

Everything from reports to overall system availability depends on having multiple copies of data. However data duplication can be difficult. Of particular concern is the use of a database replication system, such as Sybase® Replication Server®. These systems tend to be sensitive to transaction size and the use of unlogged transactions.

Understanding Transaction Size

Transaction size depends primarily on the number of operations within the transaction and the size and type of data being used.

The number of operations within a transaction should be relatively small. A transaction should equate to a single business-level operation. The two separate tasks of debiting one account and crediting the money to another form a single business operation. Either both are successful, or neither is successful. One cannot complete without the other.

Rarely does a transaction require thousands of operations, and rarely is it a single atomic operation. Generally, transactions that touch hundreds or thousands of separate data items tend to actually be many business-level operations clustered together. Separate them out into single business operations because large transactions seriously effect your system. Likewise, requiring many atomic transactions for a single business operation is detrimental to the overall aggregate system performance and leads to data corruption issues.

Accounting for Data Replication

Tuning transaction size for replication depends on the replication method used, the bandwidth available, the processing speed of machines doing the replication, and the overall topology of the system. Several techniques allow you to account for these variables at the initial stages of development.

1. Avoid large blocks of transactions. Keeping transactions to a single business operation generally avoids 80 percent of future problems. This requires a complete understanding of the business process *before* you begin to design transactions. At each step of the business process, you must ask yourself, "Must this step be completed in conjunction with the previous step? Must the next step be completed in conjunction with this one?"

2. Avoid working with large data types. Although the industry is fast incorporating many complex data types, they are still not efficient. Most replication schemes will stumble over these at worst; at best they will slow down or clog the pipeline. If you cannot avoid using complex data types, document the requirements in your initial design for future replication.

3. Most importantly, never use a transaction that cannot be detected through your replication means. For example, the phrase "Truncate table" within Sybase Adaptive Server® Enterprise is not a logged transaction and results in errors as subsequent transactions are replicated. If you use a replication system based on triggers, make sure that each transaction correctly fires the trigger and that the trigger correctly identifies which rows to change and which rows to replicate.

Anticipating Secondary Data Stores

If you are fortunate, you will have a good idea of which reports, extracts, and mass updates will be needed in your system at the time of initial design. Though these batch processes may run well at this point, is not too early to begin using a secondary data store for their operation.

As you look at the requirements for batch jobs, identify which transactions will change the data that will need to be processed. Examine these transactions and determine if they are able to make use of a secondary data store to keep track of the driving rows. If so, determine the impact of making these changes and how they affect the aggregate performance of the system.

Though this work and effort may not seem justifiable at this point in time, when you factor in the growth of the system over several years, you may find that it is very cost effective. It is important that you continually look at the life cycle cost of what you're doing.

Hot, Warm, and Cold Standby

Batch jobs are not the only place where data duplication is required. Availability often depends on data duplication as well. The three main methods of availability include :

- cold standby
- hot standby
- warm standby

Cold Standby

The most basic method of maintaining availability is cold standby. With this method, a copy of the most recent backup—typically a daily backup—is loaded onto a separate machine with a significant delay. During a loss of service, this machine can be used to restore functionality to users. However, all transactions that occurred since the backup was taken are gone. Cold standby can be an optimum choice for noncritical or offline applications and applications that receive data loads infrequently or in batch.

Hot Standby

In a hot standby system, the data is committed in both the primary and the standby simultaneously. This guarantees that the data will be in both places or neither place. The most common implementation of hot standby is "two-phase commit." However because of the difficulty in programming two-phase commit, especially in large or complex systems, this technology is often moved to middleware.

Clustering is a popular form of hot standby. With clustering, multiple CPUs are connected to a single disk. If a CPU fails, another CPU takes over using the same disk. This preserves the exact state of every transaction. When combined with mirroring or RAID technology, clustering proves to be a formidable hot standby solution. However, it only handles hardware failures, which can be a very small percentage of the types of failures you may encounter. When looking at this type of solution, consider whether or not it will meet all of your needs or if you require additional, possibly remote, copies of the data.

Warm Standby

Warm standby offers an easier version of hot standby. Application transactions are copied from one system to another after the fact. Because there is a delay, it is possible for a system to fail without having all transactions moved to its standby system. Most warm standby methods, however, have very low latency numbers; typically, they are only several seconds. The logic is that the seconds of lost transactions will be easier to make up then the amount of coding required for true hot standby. Additionally, most warm standby systems can be implemented without recoding the transactions, as is the case in two-phase commit.

Any system that must preserve transactions should maintain at least two sources of transaction recovery. For example, if the client posts a transaction to the server, a copy of the transaction should remain on the client until the backup of that transaction is made from the server. This same principle can be applied to data replication. If you have a requirement to preserve all transactions, you can still use a warm standby system. Any transactions missing when a switchover occurs can be gathered from the client. Even if you do not currently have this requirement, is a good idea to build this capability into your initial design. The ability to replay transactions from the client logs will offer benefits not only in production, but also in development, testing, and deployment.

Issues with Synchronization

No matter what method you use to replicate or duplicate data, you must address the problem of synchronization. First, you need an accurate, quick, reliable method that communicates whether or not the source and the target are in synch. Second, you need a method of resynchronizing the two systems.

Most importantly, however, you need a synchronization method that works. This means it must be part of the design plan from the beginning. As an application developer, it may fall to you to determine which business rules govern this process. You may even be required to code the tools to do it. If you are a project manager, architect, system designer, or developmental lead, this is certainly going to be within your sphere of responsibility.

If you depend on having duplicated data, whether for availability or batch processing, you need to be actively involved in understanding how that duplication takes place, the synchronization methods, and the means by which you can tell whether or not systems are in sync. No system, no matter how wonderfully built, stays synchronized forever.

Security

Security is possibly the toughest issue to bring into balance. Too much is not enough, and any at all can be quite inconvenient. Although no system is hack proof, there are certain steps you can take to protect yourself to a reasonable degree. But who decides what is reasonable? Any number of teams, from the consumer to business security to the end user, will have input on this question. And like everything else, as time goes by security needs will change.

Changes in security needs can occur from any number of events. The addition of a single data element, for instance, can change the security needs of an entire application. A merger, acquisition, or partnership can also lead to a quagmire of security issues. What is more difficult to handle, however, is the emotional urgency that comes with an increased awareness of a security threat, either through an actual incident or simply an increase in perception.

A number of basic security principles, if incorporated at the initial stages of development, can offer the most options for building an adaptable security structure. At the risk of stating the obvious, the most basic principle that you must come to grips with is, "Keep security secrets *a secret*." This may have significant impact on your development process, your code archives, and your documentation. Key people in your company, whether it is large or small, must agree beforehand about how security methods and algorithms will be kept.

The Scope of Security

As a function, security extends far beyond authorization and authentication. Security addresses business continuation on all levels, from availability to audits to building access to software. Its primary role within the organization is to assure the company that they'll be able to continue to do business no matter what the threat.

A good security organization is largely built on policies and practices. These documents dictate not just the technical aspects of security, but also how and by whom within the organization securities are handled. As a client developer, you will most likely deal with the security group on issues of authentication, authorization, intruder detection, and data protection. It is important that you understand, though, the organizational and process perspectives of this team.

You will be responsible for developing the utilities that are core to making the application usable as well as for developing the application itself.

Authorization and Authentication

Authentication is the process of establishing the user's identity. This is most often accomplished with a login and password, however a number of other methods are being developed. Single sign-on has been the Holy Grail of the computer software industry. Smart cards, digital certificates, and other technologies are increasingly becoming commonplace.

Authorization is the process of establishing what permissions the identified user has once they're in the application. Authorization goes hand in hand with enforcement, which makes sure that the user does not exceed his or her authorization. Authorization without enforcement is useless; however, in most client applications, the enforcement takes place only at the client level, leaving the server side completely open. The user who does not have permission to read a particular data item from within the application can, if they are able to access the database directly, access any information they want.

The only thing more dangerous than not having security is having only the illusion of security. Don't get fancy! There's no substitute for a simple, solid security infrastructure. Trying to simulate such an infrastructure with complex trickery only creates more loopholes for intruders to exploit.

A common practice that weakens security is hard coding passwords. This will not only create a hole for an intruder use, but it will also create a long-term nightmare. There's nothing worse than having a system in which you cannot change the password because you have lost track of all of the places that have the password hard-coded.

Intruder Detection

Even applications that do a good job of authorization and authentication often miss an obvious method for increasing security. When you have a failure to authenticate or an attempt to exceed authorization, not only should you deny access, but you should also keep track of the incident. The only way to detect breach attempts is to observe a pattern of system behavior, such as a series of failed logins coming from a particular i.p. address. If you must build your own security system for your application, keep track of both successes and failures. Use a configurable retention value to manage the size of the logs.

Data Protection

As an application developer, you can protect the most valuable asset of all: the company's data. If you encrypt key company data items, transmit, and even store them encrypted within the database, you will have achieved one of the most valuable levels of security possible.

Encapsulated small transactions are also valuable for protecting data. If the user has the permission to update a customer, that process should be encapsulated into a procedure that they are authorized to run. This procedure should affect one customer at a time. Should authorization be breached, it will take longer and will affect fewer customers. If permission is simply left at updating of customers, there's no distinction between updating one and updating a million. A security breach in this environment could easily destroy the entire customer base with a single command.

The Placement of Security

As much as is practical, centralize security. Centralization helps not only protect its secrets, but also eases administration, development, and protection, and facilitates universality. Security should be one of the fundamental business services available to applications. Enforcement of authorizations should occur at all levels of the application. Your application can and must fit into the security profile of your company. Otherwise, you will be the weak link for all of the data within the enterprise.

New Technology

The last element to plan for, but certainly not the least, is technological change. The pace of technological change is greater now than it has ever been.

The more you depend on a particular technology, the more painful it will be to replace it with new technology. Four elements create this technological dependence:

- staff skills
- degree of embeddedness
- degree of pervasiveness
- degree of customization

As an application developer, you cannot control the staff's skill set because you can't predict what the new technology will be. You also cannot control the degree of pervasiveness because it is up to each application development team to choose whether or not to use the technology. You can, however, control the degree of embeddedness and degree of customization.

Embedded and Customized Technology

Handling technology changes over time depends on the use of clean interfaces. We've talked before in this article about separating the different components of the system. The same is true with various technologies. They should be separated, encapsulated, and accessed through standard APIs.

The more you embed one technology within another, however, the more you increase technological dependence and decrease flexibility. Keeping like technologies isolated from other technology, when possible, creates a more easily movable set of building blocks.

Another way to decrease technological independence is to customize your technology. The more you move away from an out-of-the-box configuration, the more work will be required to change that technology. Any replacement technology will need the exact same customization.

Most customization occurs over long periods of time, which makes it difficult to estimate how much work is required to replace a piece of customized technology. Additionally, costs tend to be staggering.

The only exception to this rule is when you have customized proprietary technology to work with *standard* APIs. Replacing it with a technology that already uses the standard does not require additional work.

Technological dependence is not all bad. What is important is that dependence does not happen accidentally. The more conscious you are about choosing the technology you will become dependent on, the more conscious you will be about your technology choices overall. This consciousness will go long way to preventing knee-jerk reactions to the changing face of the industry. Any technology that works is not obsolete.

Managing Expectations

Adopting a new technology, even into the most isolated section of your application, is not trivial. Expect problems. Incorporate the technology on a small scale while proving concepts. Discover what is truly required to make it work.

I dictated this entire article directly into the computer using voice recognition software. The first draft of each section made very little sense. I was more that a little humbled by my shortcomings in diction. Clearly, I could benefit from a good deal of training in using this technology. Don't underestimate the impact that a technology change will have on people as well as applications. Extensive, and sometimes costly, training is usually required.

You should also stay in tune with how your company adopts new technology because that will determine to some degree your level of responsibility in the adoption process. For example, do they have a technology or architecture council, or will it be up to you to make this new thing work?

Hedging Your Bets

Your biggest win will probably be separating display logic from business logic from storage logic. The separation of client from server is essential to ensure a flexible system.

The Internet, for example, is all the rage. At its core, the technology is simply a different display model. If you have a system with a clean client/server interface and the business logic is encapsulated on the server, then you can implement a Web interface without duplicating any of the server work. The existing client and the Web client can even coexist.

Another method of laying groundwork for dealing with unexpected changes is ensuring the availability of appropriate resources. A server change may need to be made to an application, for example. But because no server programmers are available, a client programmer is assigned to the task. This resource implements the change on the client side. The result: a corrupted interface. The client and server are no longer separated. Your company and your group may not be able to avoid this situation, but if you are deliberate about making resource decisions, you can more accurately assess the impact of your existing options when the time comes to use them.

Re-Engineering

Some new technologies make truly fundamental changes to how we view the world. Whole new paradigms have been introduced with the idea of providing business services that are pervasive across applications (e.g., CORBA). Most organizations simply copy the exact same business processes they are currently using onto the new paradigm, wasting an opportunity to reconsider those business processes.

Application developers have a bird's-eye view of the business process. Through the strategic use of new technology, you can help make your company more productive and competitive. It is important, therefore, that you understand industry trends. If you understand the new technologies and the business processes that drive them, you can incorporate these process changes into your initial design even within the existing technology. Not an easy task, admittedly, but one that will put you in a position to accept the technology change when it happens.

Sometimes, though, you will have no choice but to completely retrofit an application. The only recourse at that point is the documentation of the system. I can't stress it enough: document, document, document. Not just how you coded the application.

You need to document:

- how the business process works
- what you modeled the application to simulate
- what the work flow is intended to be
- how you intended to achieve that work flow
- decisions made at key milestones
- what issues came up and how you decided to resolve them
- the key drivers for your decision making
- what you intended to do but did not have time to complete.

This is the one and only guaranteed way to make your application easier to live with later.

Putting It All Together

In this article, we've talked about the key areas that will change over time and ways to take them into account in your initial design. I am sure that you have noticed a great deal of overlap in the methods and suggestions presented. That is not accidental. Each concept interfaces with the others to form a "Web of impact." Figure one shows a representation of this. From this picture, you can see how everything is interrelated, and how a change to accommodate one area can positively impact another.

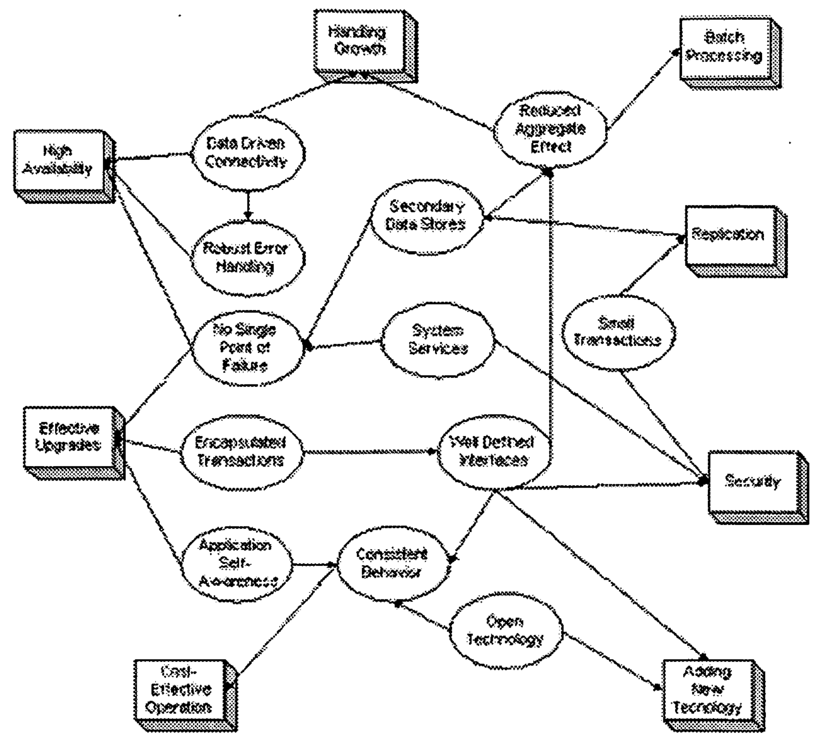


Figure 1: Web of impact.

As you go forward with application development, identify the areas that are most likely to be complicated over time according to your company's history. From there, identify the preventative measures that will impact them. You can then see which other areas will be "quick wins" with just a little more work.

I wish each and every one of you success both now and in the future. I hope that this article has helped you to build applications that you'll be able to live with later.

If you have questions, comments or suggestions please contact me at **topher.white@attws.com.**

About the Author

A 10 year I.T. veteran and database geek, Topher moved from development to operations several years ago. Serving at AT&T Wireless Services for the past several years as a DBA, Sybase® subject matter expert, operations manager, and now Internet architect, Topher has worked with large scale, mission-critical applications from a variety of perspectives. A popular speaker at the last two ISUG conferences, Topher brings a "real world, down-in-the-trenches" view, and some unique solutions, to developers worldwide. Topher lives in the greater Seattle area, where he is able to feed his ever-growing need for espresso.

Back to Top

Related Links

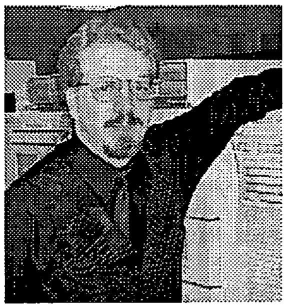
- Building a System You Can Live with Later Part 1

Document Attributes

Content Id: 1003435	**Last Revised:** March 12, 1999
Part Number: N/A	**Authorization:** Public
Focus: Technical	**Career:** Developer/DBA
Career Interest: Developer/SDN	**Document Library:** White Paper-Technical
Products: PowerBuilder	**Solutions:** Internet Applications
Expertise Level: 4-all	**Technical Topics:** Application Development

Building a System You Can Live with Later Part 1

Article by Topher White Customer Systems Operations Manager AT&T Wireless Services



By: Topher White
Customer Systems Operations Manager
AT&T Wireless Services

Introduction

If you're familiar with Murphy's Law, it should come as no surprise that it manifests itself in special ways for system design and development. During my years of large system operation, I've noticed that it rarely strikes in the initial deployment of an application, but instead waits for the system to grow and change over several years. This is because most of the application designs I've dealt with have been focused on the initial functional deliverables, with only a vague plan for what the system will eventually become.

The future is not deliberately ignored. Every application I've ever worked on has grown larger than expected, had more users than predicted, had greater demands placed on it, and was in production longer than anticipated. Under these conditions, however, Murphy strikes—hard, suddenly, and without mercy.

There are things that a PowerBuilder developer simply cannot anticipate or be responsible for; however, there is much that developers can affect. A simple decision now can have repercussions far into the future, especially if you think in terms of life-cycle costs. The initial development cost of an application is usually a fraction of the overall cost of the system during its useful life. Any decisions you can make to minimize life-cycle costs will be more important in the long run than decisions you make to cut initial development costs.

This is especially true when it comes to addressing subsequent releases. Where do you want to spend your time, energy and money? You could spend it building functional upgrades—adding new features or exploiting new business opportunities. You could also spend it fixing initial design flaws—growth and performance problems, availability and scalability issues, contention problems, or basic incompatibility.

The more you can think of the future and make decisions based on life-cycle costs, the less you'll spend fixing initial design flaws. As a result, you'll have more time and money to spend on functional upgrades, which are better for the business. This doesn't require that you predict the future. Instead, you need to look at the past. The life cycles of other applications within the organization are excellent indicators of what will happen to your application. If the requirements you have don't adequately address the future, ask for clarification. You'll be surprised at what you find out.

The eight challenges of designing for the future
As an operations' resource for many years, I've noticed eight areas in which initial system design begins to fail after a time. By accounting for the effects of these areas in your own initial design, you'll be able to accommodate their impact without much time or effort. In this first part of a two-part series, I address four of these eight areas in depth.

View the PDF file

Growth
Availability and Reliability
Upgrades
Maintenance, operation, and troubleshooting

Growth

Growth is the single biggest issue that I have seen affect systems. Growth of data affects performance; growth of users affects concurrency; either one can affect availability. Steady growth over a finite time can be handled within the constraints of the initial design, but with the tendency for applications to stay in use well beyond the predicted life span, the final sizes are often well beyond the design parameters. The common tendency for sudden, sharp growth spurts makes the problem much more difficult.

The most basic issue to address when planning for growth is the inherent scalability of an application. Although the frontend developer typically does not control scalability, he or she has a great deal of control over how easy the application will accommodate backend changes. Ideally, a client application can seamlessly incorporate changes in the number, size, and responsibility of backend servers. (See October's *Powerline Magazine* for *Economies of Scalability* by Topher White). An application constructed with dynamically data driven connectivity and data sources is able to work with any number of backend servers, whether small or large. However, there are other flaws that even a small amount of growth can expose. The most common relate to poor concurrency planning and linear data operations.

Concurrency planning
As a developer, you must be careful about making assumptions about data ownership. You only own the data that is local to you; shared data (such as a customer's account) is not owned by you, even if you make local copies. You must always assume that the data may have changed while you were working with it locally unless you have assured possession of it (i.e. it is locked on the server). Many developers get into more trouble, however, by indiscriminately locking data that being used, often disabling much more functionality than necessary. Efforts to avoid a data consistency problem then introduce a concurrency problem.

To avoid this concurrency problem as well as a data consistency problem, use a functional locking method, such as a "reserved by" field or "lock record," instead of the low-level database features while the user is working with the data. Make sure that the lock is time based so that it cannot be held indefinitely (see Figure 1). When a user begins a session, remove all locks that they may have left from previous sessions. And build small, fast, but tightly locked database transactions that are used at the last possible moment (for example, at final approval).

Account Number	Balance	Locked By	Release Time
985341	$ 100	134	8:32
985123	$ 123		

983651	$ 400	292	8:35
965342	$ 50		

The SQL for this locking method is:

```
Update CustomerAccount
set lockedBy = suser_id( ),
   ReleaseTime = dateadd(mn,getdate( ),  15)
where AccountNumber = @accountnum
and     (lockedBy is null or
        lockedBy = suser_id( ) or
  ReleaseTime < getdate( )  )
```

Figure 1: In this example, two accounts are in use. When the user begins work on the account, the row is updated with his or her user ID and a release time (15 minutes from the lock time). The client application updates the lock as necessary (in this case, every ten minutes would be good). If the client application fails to release the lock, the lock automatically expires. In the meantime, however, database access to this page is not obstructed.

Linear data operations

Reports, backups, archiving, and scans of any kind take longer as the system grows. The best way to manage this is to avoid conducting these linear operations on the entire data set. Instead, each operation should focus on only the required subset of data. I believe that time is much more valuable than space or CPU cycles. As you plan for growth, minimize the time of these linear operations as the user perceives them, even if it means doing more work elsewhere.

Make sure that reports or large queries use indexed criteria; the most commonly missed columns are date ranges. Pre-sort and pre-select data where possible. For example, keep active and inactive accounts in separate tables or copies of key data in small tables for quick access. In several systems I have worked with, we built the result set for the most common queries into a separate table at various intervals (from nightly to every five minutes) so that the client application only needed to read one small table instead of scanning all of the source data.

Another method of focusing on just the required data is to have key transactions update counters (such as "Number of Changes Today") in a small table via triggers so that the client report does not have to calculate them by reading all of the source tables.

Availability and Reliability

The aspect of a system that allows it to be used at any given time by a user is its "availability." A system that cannot be used when the user needs to use it is not available; if this happens frequently, we say that the system has poor availability. The developer controls the feature set that allows the system to be available despite external events such as hardware failures. The user expectation of the amount of time a system should be available for use is that system's "service-level requirement." For some systems, it may be 12 hours a day, five days a week; for others, it may be 24 hours a day, 7 days a week. Availability features cannot be grafted onto a system after the fact; an application must be designed from the beginning to meet its service-level requirement.

Effectively planned reliability also affects an application's flexibility. Whereas availability is a measure of the overall total amount of time spent not working (measured, for instance, in impacted user minutes), reliability is a measure of the frequency of problems. A system that has many small outages may have high availability, but it will have poor reliability.

The requirements for availability and reliability often change over time and not usually in the developer's favor. Even a relatively minor system or adjunct feature can become critical if people begin to rely on it.

This is especially true when a business process is changed to incorporate that system or feature and the former methods are lost or abandoned. Demands on the system become greater, and the requirements for system availability come to the forefront.

Because these system availability attributes cannot be easily grafted on after the fact, you should try to build them into every system from the beginning, or at least leave some standard "hook points" in the application to support the addition of features while maintaining availability should it become necessary. Though many provisions are made for availability within the IT infrastructure–redundant hardware and network, standby sites, and off-site tape storage–the client developer still wields a good bit of influence over an application's ultimate availability. Transactional consistency during a failure, for instance, is be well with the sphere of the client design.

Keys to availability and reliability

Availability is measured from the perception of the user. The user does not care if a particular machine is down as long as another has taken its place and he or she can continue working. You build availability by having an alternative if something goes wrong. The basic key to availability is in eliminating single points of failure; where you can't eliminate them, document them. For every single point of failure, find an alternative (a second server, a different network path, an alternate vendor or method) and make sure that your client application can access it dynamically.

Reducing problem duration is another key to ensuring availability, and again, finding an alternative way of providing functionality provides the answer.

Reliability, on the other hand, is built by hiding the problems from the user and automating the responses. As a rule, the construction of a reliable system depends on its availability; you can't hide the fact that a backend system has failed without an alternate system to connect to. While availability is primarily a backend responsibility that the application developer needs to be aware of, reliability is almost fully a client function.

To build reliability, you must detect errors and construct appropriate automated responses that do not involve the end user. These responses can range from simple (as in a deadlock condition) to extremely complicated (trying to handle the failure of a service for which there is no alternative). The degree to which you incorporate reliability into your application is usually a function of initial requirements and resources. However, you can go back and add more reliability at a later time *if* you have built a solid, smart, and *extensible* error-handling subsystem into the application.

Providing availability

The most standard implementation for availability is the ability to switch to another backend server or site.

Four solutions provide this availability.

> Another machine assumes the role and i.p. address of the failed machine. This solution requires no real work on the client.
>
> You can use a name resolution service (such as DNS) and then change the machine to which the name resolves. A variation on this is to have multiple machines that can be resolved in a load-balanced fashion. Again, this method requires no client changes.
>
> You can use some kind of directory service within the network architecture, such as with a CORBA infrastructure. In this case, the client program needs to make use of that service. If you have this kind of infrastructure in place in your environment, you are probably using it anyway for other services such as security and reliable messaging.
>
> In the absence of the above options, you can design the client so that it uses a list of master servers from which it can get its initial information. If the first server fails to respond, the client moves on down the list.

A significant issue for providing availability using the fourth method is the frequency with which you update the connectivity information. Simply loading the information once and then using it locally will you get you into trouble because failures generally don't occur on schedule. Be pessimistic. A failure can occur at any time and at any point during a transaction. Refreshing the local environment frequently, such as at every transaction, ensures you do not attempt to work on a failed system.

You can also extend this method to include other services or systems. Take credit checks, for example. You may want to have more than one vendor providing them. Your application should then be able to move from one vendor to another–especially if one vendor is down–without the user or client changing. Ensuring such flexibility can complicate data structure, but it is important to build a system in which changes do not need to be made on each and every client. The key to a swift resumption of business (and therefore high availability) is to take advantage of alternate systems by making changes in only one place.

Providing reliability

The most reliable systems don't simply have fewer failures. As a matter of fact, as you introduce redundancy you increase the number of failures you will have. Reliable systems mask these failures from the user. The ability to handle and respond to these failures automatically is the hallmark of reliable systems.

An extreme example of building for reliability is the n-phase commit model. In this method, all activities from a client are executed simultaneously on multiple servers. As long as some number of servers succeed in completing the transaction, the client continues. While this method of ensuring reliability is exceptionally safe, it does suffer some drawbacks–most notably being difficult to code and suffering performance challenges. A number of tools are emerging to help provide this functionality through middleware, however, so it may become much more popular.

There are simpler methods of providing reliability that can be employed when an error is received. I summarize them in four categories: retry logic, reconnect logic, transaction buffering, and partial functionality. These methods can be used in any combination and separately for different errors. You may start simply and then add more complex logic later. It's not as important that you have a lot at first, but rather that you have a structure in place to continually build it in.

Retry logic

Operations may fail due to a hard failure, such as system going down, or because of a transient failure, such as a lost network packet. The operation retry should be automatic and should repeat several times before giving up. It should also pause briefly between attempts; I have great success with a random pause time within a set range (say 100 - 500 ms) because it avoids multiple clients falling into near-synchronous retry attempts, which tends to result in collisions. Additionally, the user should not be interrupted with a dialogue box or message during the initial retry. Also, check the state of any persistent connections before the retry attempt to make sure they are still valid.

Reconnect logic

If the retry fails or a persistent connection is not valid, the application should attempt to reconnect before retrying. First, the application should attempt to connect as it was (e.g. to the same server or service). This is often the solution for a dropped database connection–due to a deadlock condition, for example.

If this reconnect fails, the application should seek an alternate connection. This search often involves moving through the initial master server list. Once a master server connection is established, the connectivity for the current transaction should be identified, the client connection to the new service established, and the transaction resubmitted. If you use complex, multistep, or layered transactions, you must remember that the entire transaction will need to be resubmitted when you reconnect–not the just the current operation.

As with other techniques for ensuring reliability, this should all occur behind the scenes without the user's knowledge.

Transaction buffering

In some cases, it is possible to buffer the failed transaction in a stable area and apply it later when the service or server is available. Buffering does present some risk because the transaction may not be valid;

also, buffering may not be possible because of a transaction dependency.

You can create two categories of operation for your system, however, and provide transaction buffering for operations without those risks. For example, imagine a banking system. A request to withdraw money must validate with the database. If the server with my account is unavailable, the transaction cannot be completed. But if I request a new ATM card and the card-issuing server is not available, the transaction can be buffered and applied later.

Transaction buffering occurs best in middleware because the client application needs to be free to accomplish other work. You may, however, process the results differently on the client when buffering occurs, possibly returning a status of "pending" instead of "completed."

Partial functionality
Nothing is more frustrating than having one failure disable the application for other operations. The final method of providing reliability requires the client application to allow nondependent operations to occur independent of each others' success or failure. This means various servers or services can be made available (or not) to the client dynamically. If the parts inventory server is not available, for example, the user should still be able to access the sales order server. If the east coast server is down, the user should still be able to access and help customers from the rest of the country. If updates cannot be posted, information should still be able to be accessed "read-only." The most important detail is that dynamic availability be accomplished without restarting the client application.

The extensible error-handling subsystem
The core of these reliability methods is having an error-handling subsystem that can be extended to add functionality–for example, adding reconnect logic to an existing retry segment. This error-handling module must have access to a great deal of state and context information. It must know, for instance, the entire transaction that is in process and what point you are at in that transaction.

Using recursion may be the way to handle this with the least amount of repeat coding. When an operation receives an error, it calls the error-handling module; the error-handling module then calls the calling module to retry. For example:

```
proc add_feature (account_no char(12), feature_code smallint)

declare context char(30)
begin
  context = account_no+string(feature_code,2)
  add_tran_hist ("add_feature",context)

  << check validity >>

  SQL retcode = ACCDB..add_feature (account_no, feature_code)
    if retcode != 0
    retcode = error_proc("add_feature", context, retcode)
    if retcode = 1111
      display_message("Transaction Will be Processed Later.")
  return retcode
end

proc error_proc (proc_name char(30), context char(30), retcode int )

if retcode = 9999      /* Network Error */
  reconnect( )

case proc_name
  "add_feature"      : add_feature_error (context, retcode)
                   : return retcode
```

```
end case

proc add_feature_error (context char(30), retcode )

account_no = substring(context, 1, 12)
feature_code = value(substring(context,13,2) )

if check_connect( ) = 0
then
   begin
      while global_retry < 10 and retcode != 0
      begin
         global_retry = global_retry + 1
         sleep (rand( ) )
         del_last_tran_queue( )
         retcode = add_feature (account_no, feature_code)
      end
    end
else
   if reconnect( ) = 0
   retcode = redo_tran_queue( )
end

global_retry = 0

if retcode != 0
   retcode = buffer_tran_queue( ) /* returns 1111 if successful */
return retcode
```

In this example, the generic error-handling module receives the procedure name and full context information, handles some generic errors, and then responds as defined according to the module. In this way, the amount of reliability can be increased module by module. The error handling for this operation (add_feature) first accomplishes a series of retries by recursively calling the add_feature function. If that fails, a reconnect attempt is made. All else failing, the transaction is buffered for later processing.

Notice that in this error-processing module we only attempt to reconnect once. You could easily build some loop processing for this. Also, we keep a transaction queue in the system–that is, a full contextual log of all of the operations in the current transaction. When the transaction is finally committed, the transaction queue is cleared. If the add_feature operation is the third operation in the transaction and we have lost our persistent connection, we will first need to perform the other two operations when we reconnect. This queue can be saved off and buffered later if appropriate.

In summary, there are four important features for the error-handling module.

The full transaction set must be available, not just the current operation.

The error handling must process by function or module.

It must receive full context for the error.

It must not interrupt the user until it has exhausted all options.

With these four goals in mind, you can create a structure with which to build reliability into a system. When combined with redundancy (the elimination of single points of failure), you will be able to meet the users' growing need for availability and robust performance.

Upgrades

No matter how excellent your initial system design and construction, over time it will change. It will have upgrades for new functionality, upgrades to reflect changes in the business, or even an occasional bug fix. With as much time and effort as goes into writing the new versions, one would think that simply getting them out onto the client machines would be simple. This simple act of deployment, however, often turns into the biggest nightmare of all, especially when there is both a client and a server component that must match.

The term "backwards compatibility" comes immediately to mind, but I find that few people can agree about what that actually means, especially in a client/server environment. At the very least, it involves ensuring that saved user settings or files must be able to be retrieved in the new version; old data must not require re-entry. But in a client/server environment, a new set of options must be addressed: new client version with old server version and old client version with new server version. In a project of any size and over the course of any significant period of time, you will need to address both of these situations. It may be a planned situation (such as the incremental cut-over of users to a new version) or accidental (such as a successful client deployment occurring at the same time as a failed server deployment). Either way, upgrades will present some challenges. To build a system you can live with later, you need to plan for these eventualities.

Initial considerations

First and foremost, components of a system must recognize each other by version. The client needs to know if the server it is talking to is the same version as the one it previously talked to, a version ahead, or a version behind. This simple exchange of information will allow you to deal with a host of unpleasant scenarios.

You can go a step a further and incorporate intelligence into this exchange. The client could, for instance, disable new functionality if it detects that the server is a version behind. The structure for recognizing this kind of exchange needs to be part of initial application design. The menu options, buttons, and function keys recognized by the client need to be version driven.

Implementation should be hidden between the components. A client need not know *how* a server will look up a stock number, only how to request that the server do so. This kind of concealment is not popular in the ODBC-enabled client world, where the logic and structure of the data is both in view of and the responsibility of the client developer. Although in the short term giving the client developer this responsibility is a cheaper and easier implementation, especially in a RAD environment, it deeply impacts the future. Essentially, it marries the client and server versions because the server implementation is in full view of, and controlled by, the client. Although manageable in a small environment, it poses logistical impossibilities in a large or highly available environment. (How do you, for instance, upgrade thousands of clients and dozens of servers in sync with one another if the system is supposed to be available 24 hours a day, 7 days a week?)

The need for decoupling the client/server version dependency has been one of the fundamental drivers behind the rapid growth of middleware solutions. Middleware can provide a degree of reprieve because it centralizes and isolates much of the client-side logic, but often the marriage of components still exists, this time between the middleware and the server. Although the middleware/server relationship is more manageable, a truly excellent solution is one in which all layers (whether you use two or three) hide their inner implementation behind a public interface.

[If these terms sound familiar, they should: they are the basic language of the Object Oriented (OO) paradigm, which is all too often restricted to programming languages. The OO paradigm is actually even more applicable at higher levels of abstraction. Viewing the client and the server as individual objects with complex public interfaces, as you will see in this section, is a key design tactic that enables a number of the techniques in this article.]

The use of encapsulated server code, such as stored procedures in Sybase's Adaptive Server® product family, is a good way of concealing the implementation between layers, regardless of whether you are building a two-tier or a three-tier client/server scheme. More importantly, if the client and the server are aware of each other's versions, the same interface can be used even if the combinations require different